H9200E

用 户 手 册



重 要 声 明

北京恒颐高科技术有限公司及其子公司保留在未通知用户的情况下，对其产品、服务及文档进行更正、修改、增减等其他一切变更的权利。在定购前，用户应获取相关信息的最新版本，并确认这些信息是完全和最新的。

在不用于商业目的的情况下，北京恒颐高科技术有限公司允许用户对其文档、产品进行复制。



公司名称： 北京恒颐高科技术有限公司
公司地址： 中国北京海淀区中关村南大街 48 号九龙商务中心 B 座 402 室
邮政编码： 100081

固定电话： 86-10-62121051/62121053/62125220/62129203
传真： 86-10-62121053
电子邮箱： marketing@hyesco.com

版 本 修 订

修订日期	版本号	修 订 描 述
2004年01月01日	001	原始版本。
2004年05月20日	002	添加 Linux 设备驱动程序的编写说明。
2005年07月18日	003	添加常见问题 Q4～Q7。
2005年07月26日	004	修正了附录 A 中 Minicom 波特率的设置，由 19200 改为 115200。
2006年03月22日	005	修正了 Uboot 的编译方法描述。
2006年06月06日	006	添加 Nand Flash 文件系统支持。



目　录

一、H9200E 简介

H9200E 是一款高性能、低价位、基于嵌入式工业控制系统的开发工具套件，可用于嵌入式工业控制系统开发人员进行 ARM 技术评估或进行嵌入式 Linux 系统设计。

H9200E 采用开放式架构设计，便于用户根据自身需求，进行各种扩展，系统的核心是 32 位 ARM920T 核的高速 ARM 处理器 AT91RM9200，AT91RM9200 是一款高性能、低功耗、低成本的嵌入式 ARM 微处理器，作为一款高性价比的 ARM 处理器，AT91RM9200 已被广泛应用于各种工业控制系统中。

H9200E 适用于嵌入式工业控制、移动计算和普适计算等场合。

1.1 板级支持包

H9200E 板级支持包是一个完整的 ARM 嵌入式系统开发平台，包含开发嵌入式工业控制系统需要的所有部件：

— H9200E 系统主板；

— H9200E 系统主板相关软件及 CD-ROM 文档；

— 9 针 RS-232C 串行电缆线；

— 输出＋5V（或＋5V、＋12V、－12V）的稳压直流电源；

1.2 硬件特性

H9200E 系统主板包含以下部件：

— ATMEL 的 AT91RM9200 微处理器，180MHz 工作频率下运行在 200MIPS，内嵌 16KB 数据 Cache 和 16KB 指令 Cache，以及 MMU；

— 4MB Flash（2M×16 位），可完全固化 Linux 内核；

— 32MB SDRAM（2×8M×16 位），可扩展为 64MB；

— 64MB NAND Flash，可存储大量数据；

— 128×8 位 IIC 存储器接口；

— 实时时钟（RTC），带后备电源接口；

— 2 个 9 针 D 型 RS-232C 串行接口，对应于 Debug UART 和 UART0；

— 1 个 RS485 接口；

— 1 个 RJ-45 10/100M 以太网接口；

— 2 个 GPIO 接口，可由用户编程为输入或输出，带 LED 电平指示，对应于 PC14、PC15 口；

— 20 针 JTAG 接口；

— 1 个 USB 主口、1 个 USB 从口；

— ＋5V、＋12V、－12V 电源输入接口，方便用户的系统扩展；

— 完全的系统总线驱动，方便用户系统扩展；

— 完全的系统总线扩展接口，采用 PC104 连接器，便于用户进行系统扩展；

1.3 软件特性

— GNU 工具链；

— 完全移植到 H9200E 的 Linux 内核源码，Nand Flash/Nor Flash 文件系统支持；

— 基于 Linux 的设备驱动例程；

— 完全移植到 H9200E 的 uC/OS II 源码，适合于硬实时（Hard Real-Time）工业控制应用；

— 完整的例程，使用户快速上手，并方便组建自己的应用；

— 完全移植到 H9200E 的 Uboot 源码，方便用户定制及系统开发；

— 已编译完毕的二进制代码：Linux、uC/OS II、Uboot 等；

— 全部 Protel 格式的硬件原理图、所用器件库、PCB 封装库等；

二、H9200E 硬件描述

H9200E 是一款基于 ARM 的完整工业控制应用开发平台，包含常用的核心功能，是一款功能强大、配置灵活的嵌入式工业控制应用技术的评估与开发工具，系统主板外观如下图所示：

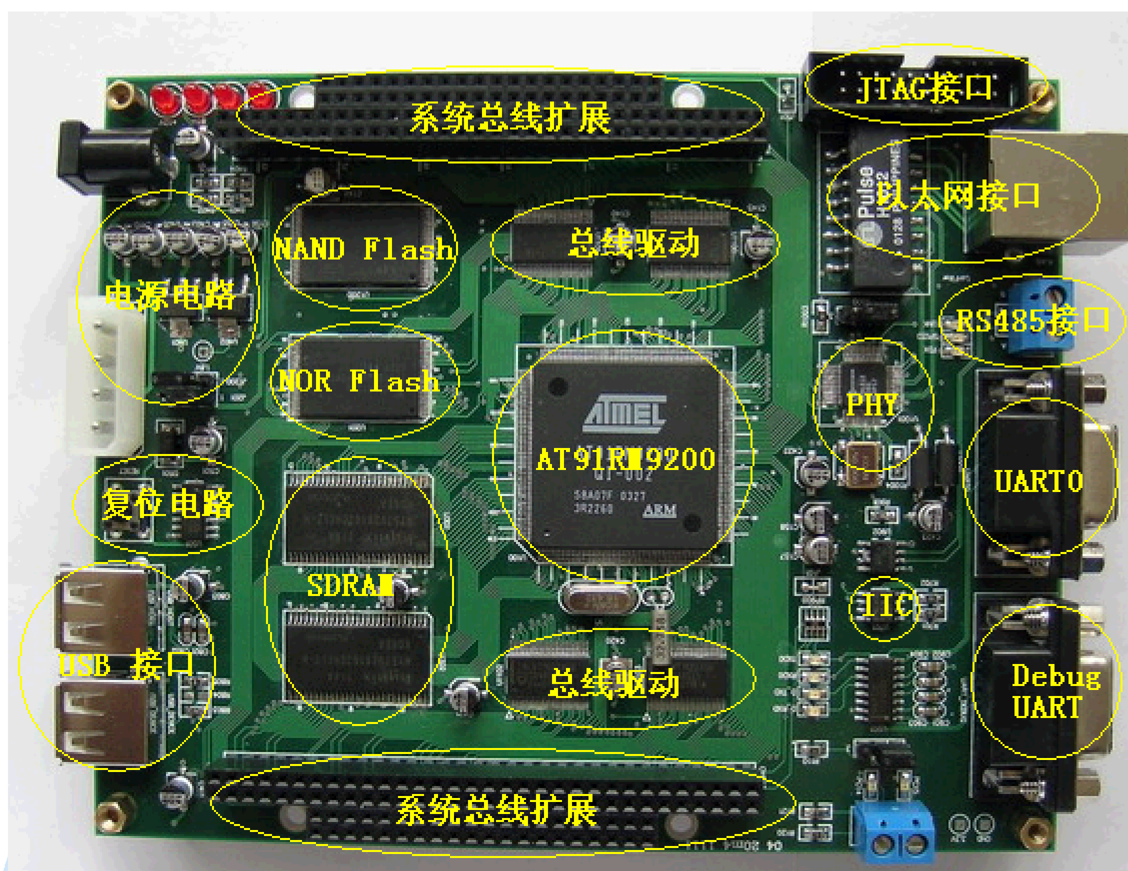


图 2.1　系统主板外观及功能说明

其主要功能模块描述如下：

2.1　　ATMEL 的 AT91RM9200 微处理器

AT91RM9200 微处理器是一款由 ATMEL 设计生产的嵌入式 ARM 微处理器，采用 208 脚 PQFP 封装，内含一个 ARM920T 核和以下片内外围，其主要特性如下：

- **ARM920T ARM-Thumb 微处理器核：**
  - 在180MHz的工作频率下运行速度为200MIPS；
  - 内嵌16KB的数据Cache，16KB指令Cache，写缓冲区；
  - 全功能的MMU（Memory Management Unit）；
  - 内嵌Debug通信通道的Emulator；
  - 嵌入式Trace Macrocell （仅限于256-ball BGA 封装模式）；

◆ **片内存储器：**
- 内嵌16KB SRAM和128KB ROM，方便用户调试；

◆ **外部总线 EBI 接口：**
- 支持SDRAM、Static Memory、Burst Flash、Glueless Connection to CompactFlash®、SmartMedia™ 和NAND Flash；

◆ **功能强大的片内外围电路：**
- 增强型的时钟产生器和电源管理控制器；
- 带有两个PLL的片内振荡器；
- 慢速的时钟操作模式和软件电源优化能力；
- 四个可编程的外部时钟信号；
- 包括周期性中断、看门狗和第二计数器的系统定时器；
- 带有报警中断的实时时钟（RTC）；
- 调试单元，支持两线UART的Debug调试通道；
- 带有8个优先级、可单个屏蔽中断源、Spurious中断保护的先进中断控制器；
- 7个外部中断源和一个快速中断源；
- 四个32位的PIO控制器，可达122个可编程I/O引脚（每个都有输入控制、可中断及开路的输出能力）；
- 20通道的外部数据控制器(DMA)；

◆ **10M/100M 网卡：**
- 传输媒体独立接口（MII）或精简传输媒体独立接口（RMII）；
- 集成28字节的FIFOs和直接用于收发DMA通道；

◆ **双 USB 2.0 主口(12 M-bits/秒)：**
- 两个在片的收发器（208脚的PQFP封装只有一个）；
- 集成的FIFOs和DMA通道；

◆ **一个 USB 2.0 从口(12 M-bits/秒)：**
- 一个在片的收发器；2KB可配置的集成FIFOs；

◆ **多媒体卡接口（Multimedia Card Interface，MCI）：**
- 支持自动协议控制和快速自动数据传输器；
- 兼容MMC卡和SD存储卡，最多可支持两个SD存储卡；

◆ **三个异步的串行控制器（Synchronous Serial Controllers，SSC）：**
- 每个收发器均有的独立的时钟和帧同步信号；
- 支持I2S模拟接口，支持时分多路复用；
- 支持32位高速连续数据流发送；

◆ **四个同步/异步收发器 (USART)：**
- 支持ISO7816 T0/T1标准智能卡；
- 支持软、硬件握手功能；

- 支持RS485传输、速度可达115 Kbps的红外传输；
- USART1带有全功能Modem控制线；

- **主/从串行外围接口（SPI）：**
  - 8/16位可编程数据长度， 4个外围芯片选择；

- **两个 3 通道，16 位的定时器/计数器（TC）：**
  - 3个外部时钟输入， 每个I/O通道有两个可以复用的I/O口；
  - 双PWM生成，捕获/波形模式，向上和向下计数兼容；

- **两线接口（Two-wire Interface，TWI）：**
  - 以主模式支持所有两线的Atmel EEPROMs设备；

- **IEEE 1149.1 JTAG 边界扫描接口；**

- **供电电源：**
  - 1.65V 到1.95V for VDDCORE, VDDOSC and VDDPLL
  - 1.65V 到3.6V for VDDIOP (外设I/Os) and for VDDIOM (内存I/Os)

- **可选择 208 脚 PQFP 和 256 脚 BGA 封装；**

2.2 Flash 存储器

H9200E 系统主板包含 4MB（可替换为 2MB）Flash 存储器，内部存放启动代码、Linux 内核、用户程序等。Flash 存储器的数据宽度为 16 位，可通过跳线（J201）选择映射到 AT91RM9200 的 Chip Select 0 或 Chip Select 7。

2.3 SDRAM 存储器

H9200E 包含 32MB SDRAM，作为程序的运行空间。由两片 16 位数据宽度的 SDRAM 存储器并联为 32 位数据宽度的 SDRAM 存储系统，并映射到 AT91RM9200 的 Chip Select 1。

AT91RM9200 微处理器内部的 16KB 的 SRAM，通常被配置为堆栈区以提高系统性能。

2.4 NAND Flash 存储器

H9200E 包含 64MB（可替换为 32MB）的 NAND Flash 存储器，作为系统的数据存储器。NAND Flash 存储器的数据宽度为 16 位，映射到 AT91RM9200 的 Chip Select 3。

2.5 异步串行通讯口

H9200E 外接两个 UART（Universal Asynchronous Receiver/Transmitter），Debug UART 和 UART0，Debug UART 用于软件调试与系统开发，UART0 可完成与 PC 及 Modem 的通信。

RS-232C 接口定义见下表：

引脚	名称	功能描述
1	DCD	数据载波检测
2	RXD	数据接收
3	TXD	数据发送
4	DTR	数据终端准备好
5	GND	地
6	DSR	数据设备准备好
7	RTS	请求发送
8	CTS	清除发送
9	RI	振铃指示

2.6 RS485 接口

AT91RM9200 的 USART 均支持 RS485 通讯模式，H9200E 在 UART1 外接了一片 RS485 总线收发器，用于支持 RS485 通讯。

2.7 10M/100M 以太网 MAC

以太网 MAC 是 OSI 参考模型中界于物理层（PHY）与逻辑链路层（LLC）之间的 MAC 子层的硬件实现，以太网 MAC 支持 MII（Media Independent Interface）和 RMII（Reduced Media Independent Interface）模式的数据传输。

H9200E 外接了一块物理层芯片 DM9161，用于支持以太网通讯。

欲详细了解 AT91RM9200 的以太网 MAC，可参考 ATMEL 关于 AT91RM9200 的用户手册。

2.8 USB 接口

利用 AT91RM9200 片内集成的 USB 通讯控制器，H9200E 外扩了一个 USB 主口和一个 USB 从口，用于支持 USB 设备的读写；

2.9 IIC 存储器

H9200E 外接了一个 128 字节的 IIC 存储器 AT24C01，用于存储 IP 地址、MAC 地址及系统加密等。

鉴于系统已经很好的支持了 Flash 文件系统，用户也可以以文件的方式将系统信息存储在 Flash 中，而不需要使用 IIC 存储器。

2.10 系统总线驱动

为方便系统的扩展，增加系统总线的驱动能力，H9200E 对所有的数据总线、地址总线和部分控制总线进行了驱动，经过驱动的总线可以方便的外接大量外设，便于用户组建复杂的嵌入式工业控制系统。

2.11 跳线及 LED 显示器

1、跳线

H9200E 系统主板使用了几组跳线进行功能选择：

J100 为启动模式选择（BMS）：当跳线 J100 短接在 1－2 时，系统复位后从 Flash 启动，当短接在 2－3 时，系统复位后从 AR91RM9200 内部的 ROM 启动。更详细的连接方法可参考系统主板原理图。

找到 J100 在 H9200E 上的位置（在靠近串口的地方），在以后的操作中我们经常会用到的。

J201 为 Flash 芯片的片选信号：当跳线短接在 1－2 时（系统默认），Flash 映射到 AT91RM9200 的 Chip Select 0，短接在 2－3 时，Flash 映射到 AT91RM9200 的 Chip Select 7。更详细的连接方法可参考系统主板原理图。

注意 :为方便用户使用，有的版本已将该跳线取消，而直接将 Flash 映射到 AT91RM9200 的 Chip Select 0。

J500 为 JTAG 选择：当跳线短接在 1－2 时（系统默认），系统选择 JTAG 调试模式，短接在 2－3 时，系统选择嵌入式 ICE 模式。更详细的连接方法可参考系统主板原理图。

注意：为方便用户使用，有的版本已将该跳线取消，而直接将系统选择 JTAG 调试模式。

J1200 为 NAND Flash 的片选信号：当跳线短接在 2－3 时（系统默认），将 NAND Flash 映射到 AT91RM9200 的 Chip Select 3。更详细的连接方法可参考系统主板原理图。

注意：为方便用户使用，有的版本已将该跳线取消，而直接将 Nand Flash 映射到 AT91RM9200 的 Chip Select 3。

2、LED 显示器

H9200E 系统主板带有多个 LED 显示器，用于指示系统的工作状态：

位于系统主板电源电路部分的 4 个 LED 为电源指示，分别指示 12V、5V、3.3V 和－12V 电源电压，当系统接通电源时，相应的 LED 亮。

标识为 PC14 和 PC15 的 2 个 LED 为 GPIO 口 PC14 和 PC15 的电平指示，分别指示其输入/输出电平状态。

标识为 D_RXD、D_TXD、RXD0、TXD0 的 4 个 LED 为 Debug UART 和 UART0 的工作指示，当 UART 进行数据通讯时，LED 闪烁。

标识为 FXD、SPEED 和 LINK 的 3 个 LED 为以太网工作状态指示。

更详细的连接方法可参考系统主板原理图。

2.12 JTAG 接口

为方便用户的调试，H9200E 设计有 20 针标准 JTAG 接口。20 针接口与 ARM Multi-ICE 接口兼容。

2.13 复位电路

H9200E 的复位电路可以完成上电复位和系统运行时按键复位的功能。在系统正常运行时，当用户按下复位按钮 RESET 时，AT91RM9200 进入复位状态，松开按钮，AT91RM9200 进入正常工作状态。

2.14 电源电路

H9200E 的电源电路可以提供系统工作所需的 3.3V 和 1.8V 的稳压直流电源。

系统主板有两个电源输入接口，标识为 CON401 的接口输入为 5V 稳压直流电源，标识为 CN400 的接口输入为 5V、12V 和－12V 稳压直流电源，用于系统的扩展。

注意：过高的输入电压会造成系统的永久损坏！



三、快速上手

通过前面的内容，你可能已经初步了解了H9200E的概况，接下来的问题是：我如何尽快将H9200E玩转，并进行我自己的特定应用系统开发？

以下的内容将回答这个问题！

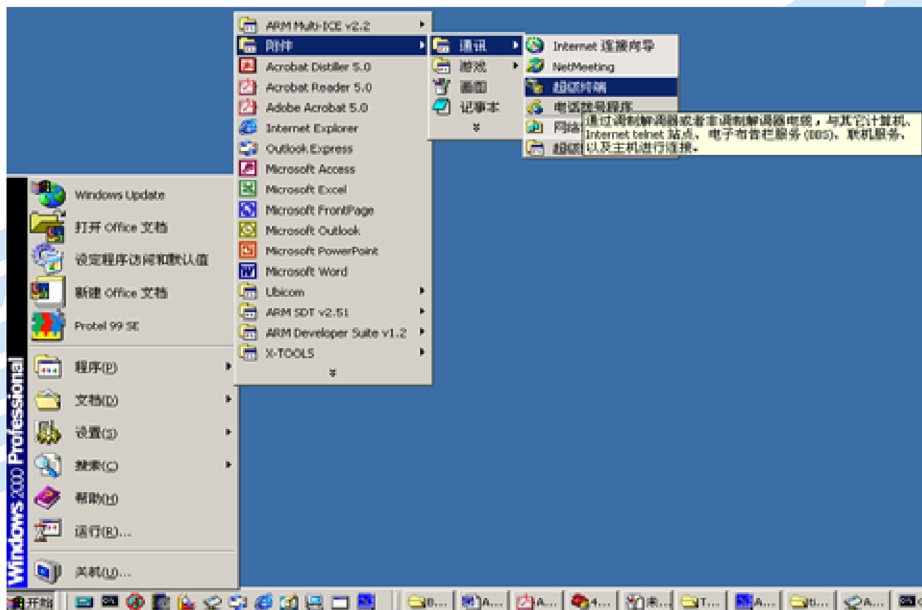
当你拿到H9200E系统时，摆在你面前的是一大堆开发资料，如果你已经开发过类似的系统，这部分内容你可以略过，去文档CD中找你关注的内容，如果你对ARM系统开发还不太熟悉，你会不知所措，不要紧，这也是正常现象，这一章会让你消除你的恐惧心理，逐步对你的产品开发发生兴趣。

这部分我们不讲关于H9200E原理性的内容，只是去熟悉它，初步了解它：

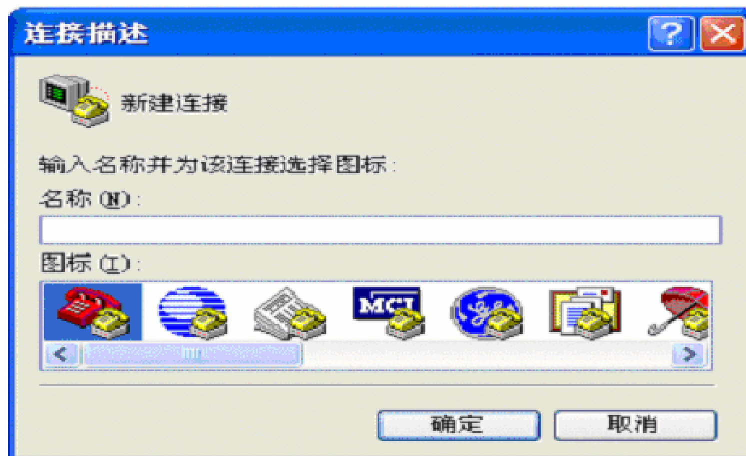
OK！Let's go…

3.1 设置超级终端

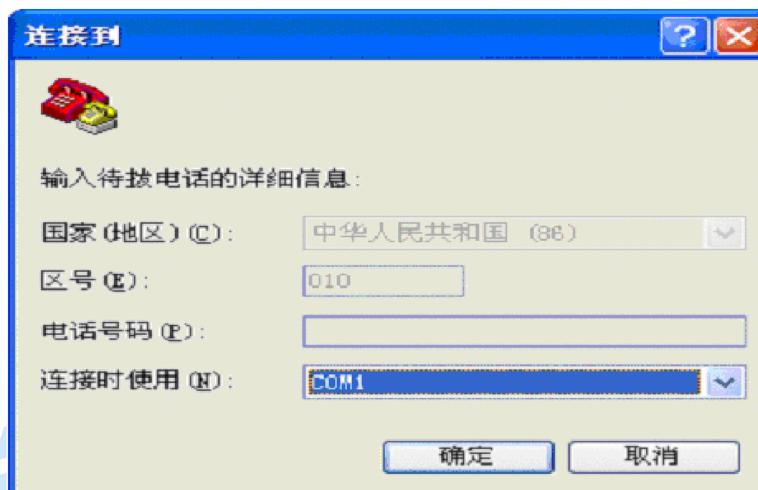
在Windows自带的应用程序中有一个超级终端，其打开方法为：开始→程序→附件→通讯→超级终端，如下图所示：



打开后将出现如下界面：



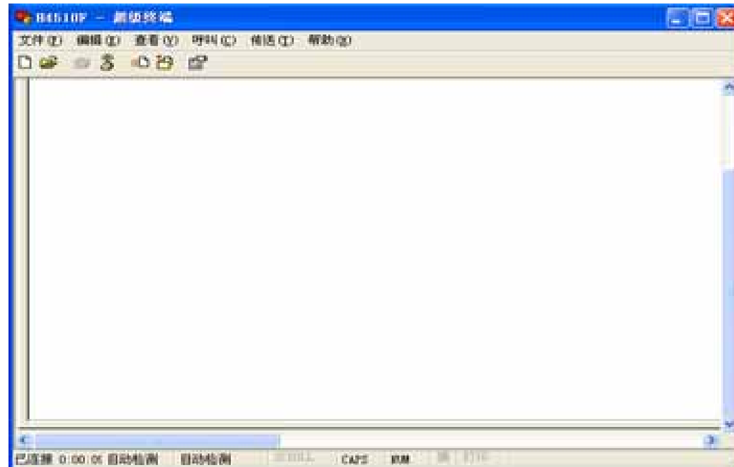
你可以输入一个自己喜欢的名字，然后点击确定按钮，就会出现如下界面：



设置连接时使用为串口1（COM1）(也可使用COM2，依实际接法为准),点击确定按钮进入下一步：



请按上图中的设定值进行设定：设置串口每秒位数115200，数据位：8，奇偶校验：无，停止位：1，数据流控制：无。点击确定按钮，即可进入下图所示的超级终端：



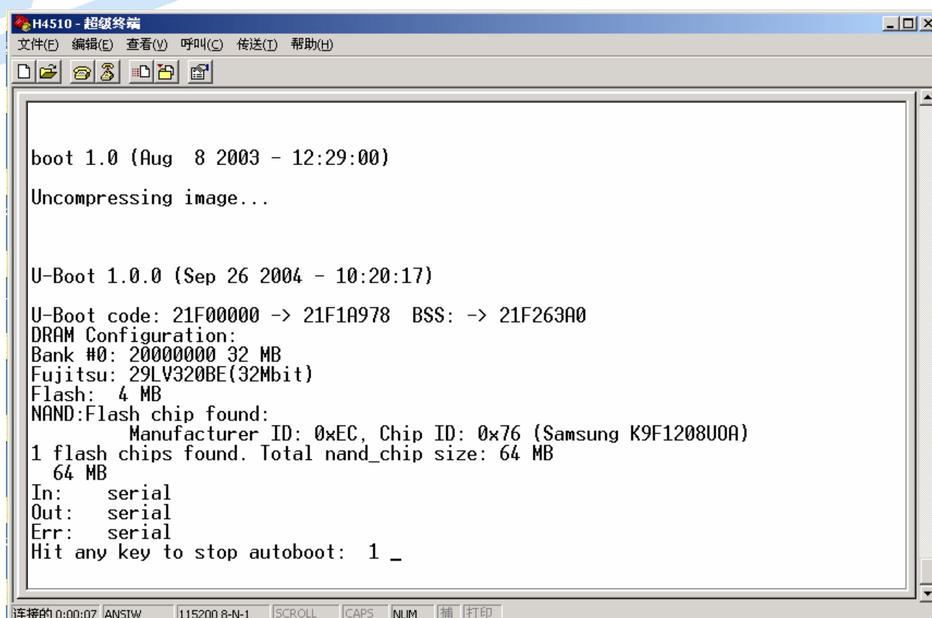
3.2　硬件连接

为H9200E系统主板连接好电源和通讯线路：首先，把随系统附带的标准串口线的一端连接到PC机的串口上（COM1或COM2的选择要和超级终端的设置相一致），串口线的另一端接到H9200E的串口UART_DEBUG上（靠角上的那一个）。接下来为H9200E接好电源，当系统上电时，电源插座旁的LED指示灯会发光。

注意：上电之前一定要看好，你使用的电源是不是符合系统的电压要求，否则就要冒烟的哦！

3.3　操作 Uboot

将H9200E的跳线J100（用于启动模式选择）的1－2短接，在接通系统电源后，超级终端中就会看到系统主板固化代码的启动信息，如下图所示：



当程序运行到此处时，系统有一小段等待时间，按下任意健，系统会出现提示符，否则系统会自动运行固化的Linux操作系统。

```
U-Boot 1.0.0 (Sep 26 2004 - 10:20:17)

U-Boot code: 21F00000 -> 21F1A978  BSS: -> 21F263A0
DRAM Configuration:
Bank #0: 20000000 32 MB
Fujitsu: 29LV320BE(32Mbit)
Flash:  4 MB
NAND:Flash chip found:
                 Manufacturer ID: 0xEC, Chip ID: 0x76 (Samsun
1 flash chips found. Total nand_chip size: 64 MB
  64 MB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
Uboot>
连接的 0:03:46  ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

键入“ help ”或“ ？ ”，出现Uboot所支持的全部命令：

```
fsload  - load binary file from a filesystem image
go      - start application at address 'addr'
help    - print online help
imls    - list all images found in flash
loadb   - load binary file over serial line (kermit mode)
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nand    - NAND sub-system
nboot   - boot from NAND device
nm      - memory modify (constant address)
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
tftpboot- boot image via network using TFTP protocol
version - print monitor version
Uboot> _
连接的 0:11:40  ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

OK！你可使用 Uboot 的命令来进行操作了，恭喜你！

3.4 Uboot 是个什么东东啊？

Uboot 是一个非常复杂的东东！

Uboot是一段非常复杂的代码！

Uboot是一个通用的BootLoader程序，非常适合于使用在嵌入式系统中，目前已广泛应用于X86、ARM、MIPS、PowerPC等架构的系统中。

1、Uboot 能干啥呢？

Uboot和其他任何BootLoader都是一样的，主要是对系统进行初始化、系统引导、Flash操作等功能。

H9200E的Uboot主要是对系统的硬件进行初始化，包括时钟和PLL、定时器、调试串口（Debug UART）等等，同时还实现了一个TFTP Client，支持网络文件传输等，当系统不能正常启动时，需要使用Uboot的相关功能进行初始化操作。

总之，有了Uboot，我们可以在主机的超级终端通过调试串口和H9200E进行通信和系统设置。

2、UBOOT 常用命令

再来看看Uboot的常用命令和用法吧，这对系统的开发是一定的帮助。

1、loadb

功能：通过串口kermit协议下载文件到RAM

语法：loadb [ram地址]

2、cp.b

功能：从RAM拷贝文件到Flash

语法：cp.b [ram地址] [flash地址] [十六进制文件大小]

3、erase

功能：擦除Flash

语法：erase [起始地址] [结束地址]

注：地址必须是Flash sector的边界地址

4、setenv

功能：设置环境变量

语法：setenv [变量名] [命令]

5、go

功能：跳转到一个地址处执行程序

语法：go [地址]

注：地址是所有地址范围内的地址值

6、bootm

功能：启动操作系统

语法：bootm [ram地址]

还有其它的UBOOT命令，可以使用帮助命令在线查看：UBOOT>？

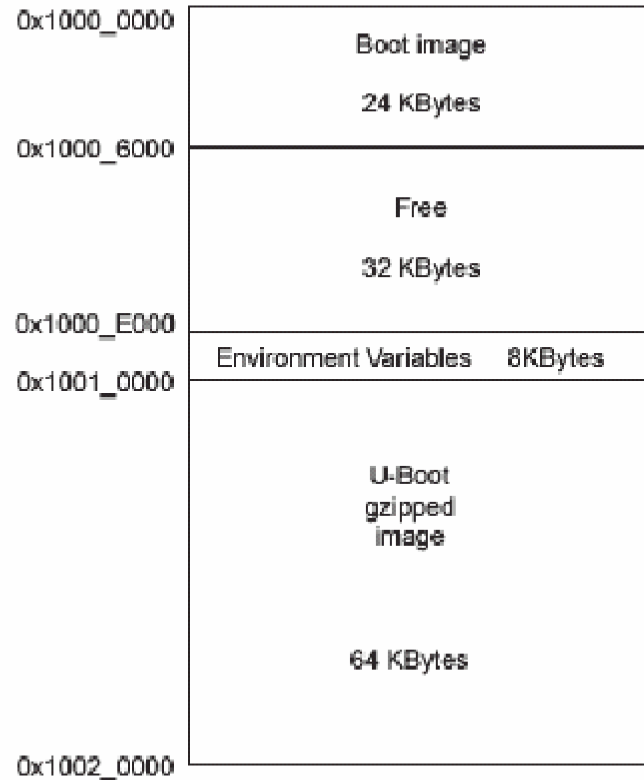
3、有关 Uboot 的资源

我们提供如下用户可能用到的有关Uboot的资源：

- Uboot、Boot和Loader 的源代码；
- 已编译的Loader.bin、Uboot.bin、boot.bin和U-BOOT.gz文件；
- Uboot的使用说明；

4、Uboot 的代码组织

Uboot的代码组织如下：



最上端的部分Boot image必须驻留在AT91RM9200 的Flash起始地址处，即0x1000，0000；当跳线J100的1－2短接时，只要系统上电或复位，程序就会从上述地址运行。

接下来的32KB为暂时未用的存储区，便于Uboot的功能扩展；

再往后的8KB为Uboot的环境变量区，主要用于存放系统需要的参数；

5、我要重新编译 Uboot

重新编译UBoot对于深入的系统开发是必须的，当然，最好详细阅读Uboot 的readme、makefile和源代码。

UBoot的编译过程需要PC＋Linux平台的支持，关于如何建立基于PC＋Linux平台的编译环境，我们在其后相关的部分描述。

Uboot的编译方法详见附录B。

OK！掌握的不错，先休息一会儿吧。

四、使用 ADS 开发 H9200E

本章我们使用ARM公司的集成开发环境ADS来进行简单应用程序的开发。

ADS是用于基于ARM的系统开发工具，是SDT的升级版本，功能更强大一些。

ADS 主要由三个部分组成：Multi-ice Server（连接工具，用于识别ARM 内核）、Codewarrior（集成编辑、编译和链接工具）和AXD（调试工具）。

以下我们介绍ADS的安装与程序调试方法。

4.1 开发环境的硬件连接

开发环境的硬件连接如下图所示。

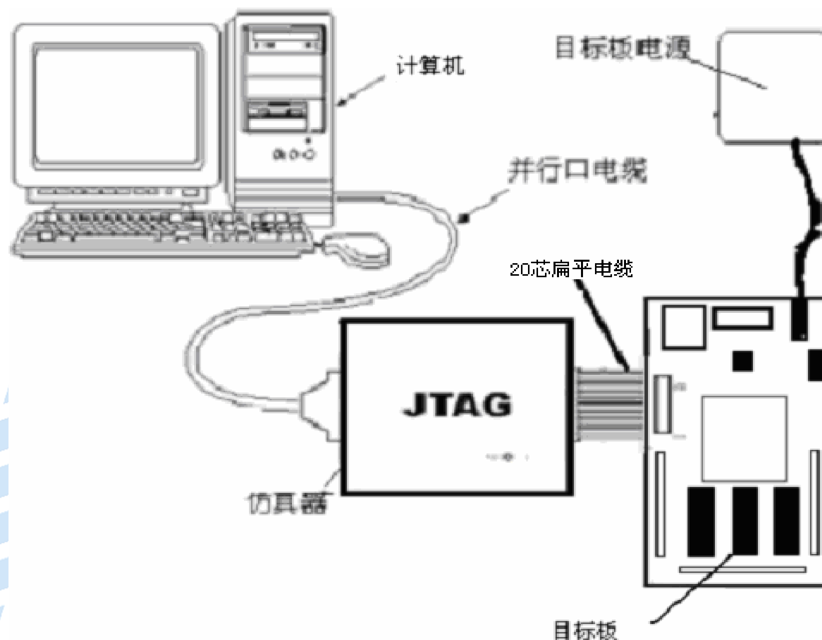


图4.1 开发环境的硬件连接

将系统配备JTAG仿真器的一端与计算机的并口连接，另一端通过20芯的扁平电缆与H9200E的JTAG接口相连，然后给H9200E上电。

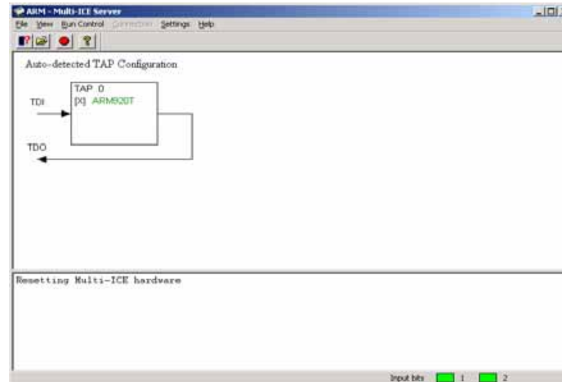
注意H9200E的跳线J500此时应是1－2短接，选择JTAG调试模式，有的主板不包含跳线J500，则已自动选择JTAG调试模式。

4.2 使用 ADS 进行程序开发

我们使用 ADS1.2 集成开发环境进行程序的开发，ADS1.2 及仿真器驱动程序的安装请参考相关文档。

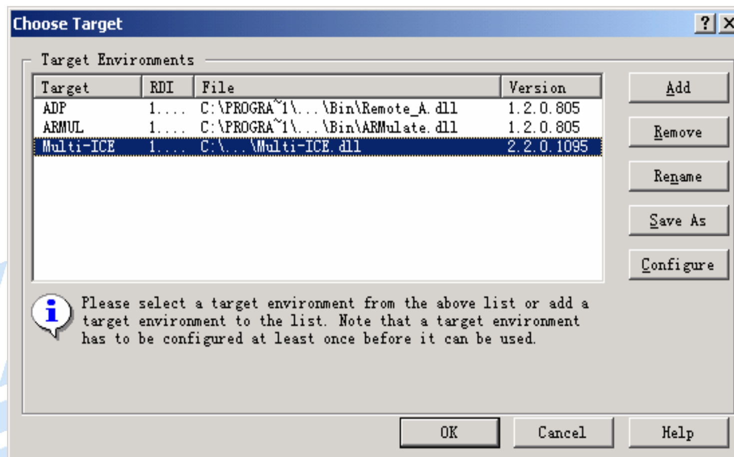
1、连接仿真器

打开Multi－ICE Server，建立与仿真器的连接，在File菜单下，点击load configure，装入ARM 的配置文件ARM92T.cfg（配置文件的位置请参见光盘说明）：



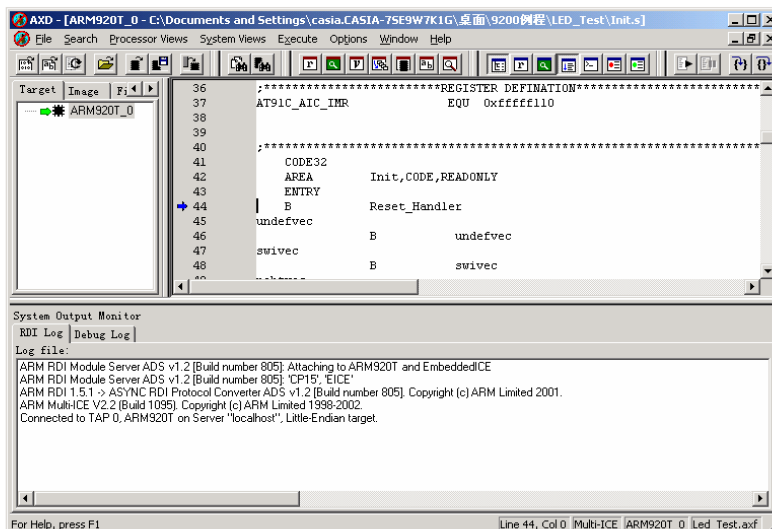
2、建立 AXD 与仿真器的连接

打开AXD，选择菜单Options－configure Target，选择Multi－ICE：



3、载入可执行映象文件运行

选择菜单 File－load image，载入光盘例程 LED_Test.axf：



此时，可以进行单步、断点运行程序，同时可查看变量、寄存器、存储器的值，进行程序调试。

如全速运行该例程，可以看到 H9200E 的两个 LED 指示灯 PC14、PC15 被交替点亮，表明程序已经在正常运行。

4.3 在 SDRAM 中运行程序

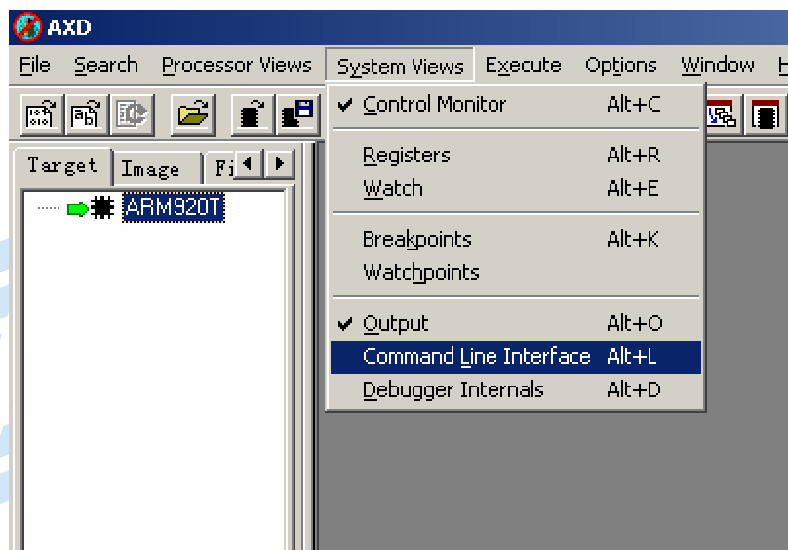
对于 AR91RM9200 这款芯片，它的片内有 16KB 的 SRAM，起始地址固定为 0x20,0000，在任何情况下都是可以访问到的，以上的例程，我们在编译指定连接器地址时，就定位到这个地址，因此程序是在片内的 SRAM 中运行。

但当用户的程序较大时，片内的 SRAM 就可能放不下了，需要放到片外的 SDRAM 中运行，SDRAM 的起始地址固定为 0x2000,0000，但 SDRAM 需要初始化，不能直接读写，因此，在访问 SDRAM 之前，必须进行初始化，可按如下方式操作：

1、 将光盘中的 ads9200.txt 文件拷贝到 C 盘根目录，该文件用于初始化 SDRAM。

2、 按上述步骤连接 H9200E 和仿真器，打开 AXD。

3、 在 AXD 中调出命令行窗口，如下图：



4、 在命令行窗口中输入命令：

>obey　c:\ads9200.txt

此时，控制 SDRAM 访问的寄存器被初始化，SDRAM 可以被访问，打开存储器窗口，将地址设定为 SDRAM 的地址：0x2000,0000，如果可以修改存储器的内容，则表明 SDRAM 已可以正确访问。

ARM920T - Memory Start addr 0x20000000

Tab1 - Hex - No prefix | Tab2 - Hex - No prefix | Tab3 - Hex - No

Address	0	4	8	c
0x20000000	AAAAAAAA	55555555	E7FF0010	E800E800
0x20000010	E7FF0010	E800E800	E7FF0010	E800E800
0x20000020	E7FF0010	E800E800	E7FF0010	E800E800
0x20000030	E7FF0010	E800E800	E7FF0010	E800E800
0x20000040	E7FF0010	E800E800	E7FF0010	E800E800
0x20000050	E7FF0010	E800E800	E7FF0010	E800E800
0x20000060	E7FF0010	E800E800	E7FF0010	E800E800
0x20000070	E7FF0010	E800E800	E7FF0010	E800E800

5、 重新编译以上例程，将连接器的起始地址定位为 0x2000,0000，对应于系统的 SDRAM，然后就可以下载运行了。

以上我们简述了如何开发在H9200E上运行程序，你可能还有很多的问题需要搞清楚，比如ADS 的使用、各种文献和硬件的熟悉与阅读等，但只要你能完成以上的步骤，剩下就看你的了，对于这些问题，你需要慢慢的熟悉，仔细阅读相关文档和资料。

祝贺你，现在，你真正已经开始ARM的开发了。

五、深入开发 H9200E

当你完成了自己的应用程序的调试，想看看程序在 H9200E 上的运行效果，如何将自己的应用程序下载到 H9200E 上并运行呢？或者，你自己已经设计了基于 AT91RM9200 的硬件系统，如何往系统下载程序呢？

以下的内容将回答这个问题。

5.1 如何对裸系统进行操作

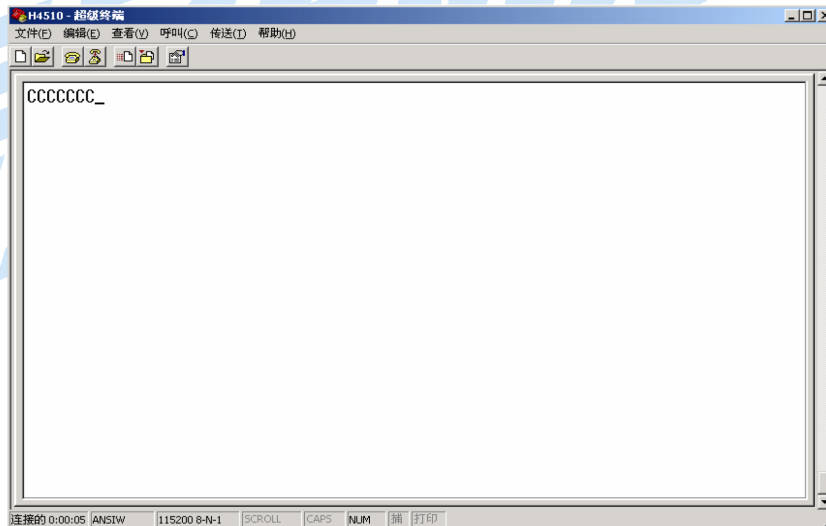
当 H9200E 由于某种原因不能启动，如固化在 Flash 中的程序被无意中擦除，或者你自己已经设计了一款基于 AT91RM9200 的硬件系统，如何下载程序到硬件系统呢？此时，你需要运行 AT91RM9200 片内 ROM 的程序，来完成以上的操作。

请按以下的步骤进行，你会发现整个过程如此简单！

注意：在以下的操作过程中，你会使用到 4 个文件：boot.bin、loader.bin、u-boot.bin 和 u-boot.gz，这 4 个文件位于 CD：Software\uboot\bin；

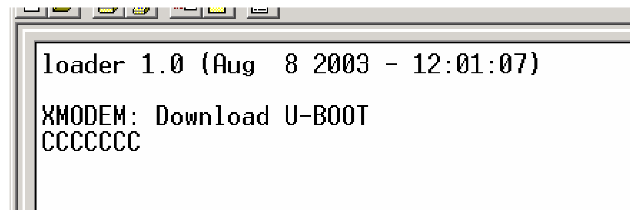
1、启动片内 ROM 的程序

将H9200E的跳线J100的2－3短接，然后复位系统，打开超级终端（115200、8、无、1、无），在超级终端会出现“ CCCCCCCC….. ”，此时AT91RM9200片内ROM的程序已开始运行，等待你的下一步操作。

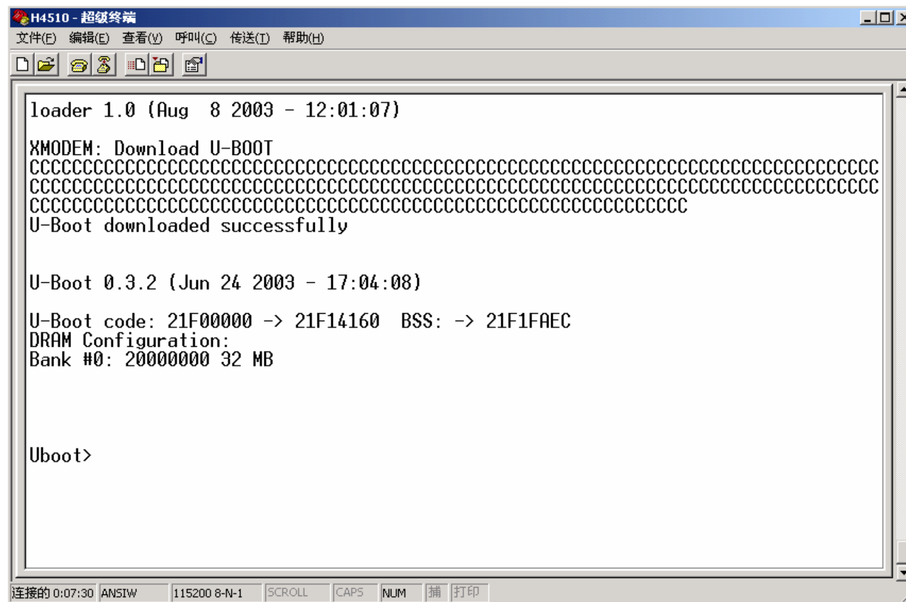


2、下载 Uboot 到 SDRAM 运行

在超级终端，使用Xmodem协议，发送loader.bin文件，然后超级终端会出现下载Uboot的提示，并继续出现“ CCCCCCCCC……… ”



继续使用Xmodem协议，发送uboot.bin文件，此时uboot.bin被发送到系统的SDRAM中，发送完毕后uboot开始运行，显示>U-Boot的提示符：



3、 擦除 Flash

在对Flash进行烧写之前，需要将其擦除：

Uboot>protect off all ；去掉Flash的扇区写保护

Uboot>erase all ；擦除Flash的所有扇区

4、 烧写 Boot.bin 到 Flash

在Uboot提示符下键入命令：

Uboot>loadb 20000000 ；将文件发送到系统的SDRAM中

然后在超级终端使用Kermit协议，发送文件boot.bin，发送完毕后，键入以下命令：

Uboot>cp.b 20000000 10000000 5fff ；将发送到SDRAM中的数据写入Flash

Uboot>protect on 10000000 10005fff ；对写入Flash的内容进行写保护

5、 烧写 Uboot.gz 到 Flash

在Uboot提示符下键入命令：装入Uboot.gz

Uboot>loadb 20000000 ；将文件发送到系统的SDRAM中

然后在超级终端使用Kermit协议，发送文件Uboot.gz，发送完毕后，键入以下命令：

Uboot>cp.b 20000000 10010000 ffff ；将发送到SDRAM中的数据写入Flash

Uboot>protect on 10000000 1001ffff ；对写入Flash的内容进行写保护

至此，你已经完成了将Uboot烧写到Flash的工作，关闭电源，将H9200E的跳线J100的1－2短接，上电复位后，超级终端显示Uboot的启动信息。

你瞧，就这么简单。

5.2 如何将你的程序写入系统并运行

当你完成以上步骤，系统就回复到正常的运行状态了，接下来的问题是：如何将自己已调试好的应用程序，下载到H9200E运行，形成一个独立的嵌入式应用呢？

以下我们用两种方式来达到上述目的。

1、在SDRAM中运行你的程序

STEP1：使用ADS调试环境调试你的应用程序，然后生成.bin格式的可执行文件；

STEP2：将H9200E的跳线J100的1－2短接，选择从片外FLASH启动UBOOT；

STEP3：键入命令Uboot>loadb 20000000，然后在超级终端使用Kermit协议，发送文件.bin文件到系统的SDRAM；

STEP4：键入命令Uboot>go 20000000

你的程序就会运行。

当然，采用这种方式，程序是在SDRAM中运行，当系统复位或调电后，你的程序就丢失了，如何将应用程序固化在系统中，当系统上电或复位时，就能自动运行呢？

以下的方法就可以实现，但步骤会稍微麻烦一些，要有耐心哦！

2、将程序固化在H9200E系统中

STEP1：首先设置Uboot的启动参数（环境变量），以便Uboot能自动将你的程序从Flash中搬运到SDRAM中高速运行，步骤如下，注意不要敲错：

Uboot>setenv usr_prg cp.b 10300000 20000000 ×（×是用16进制数表示的用户程序的大小，如ffff，表示用户程序小于64KB）

Uboot >setenv run_prg go 20000000

Uboot >setenv bootcmd run usr_prg\；run run_prg

Uboot >saveenv

STEP2：然后使用Uboot将程序烧写到系统的Flash中，键入如下命令；

Uboot>protect off all ；去掉Flash的扇区写保护

Uboot>erase 10300000 103fffff ；擦除Flash的用户区域

Uboot>loadb 20000000 ；下载.BIN文件到SDRAM

Uboot>cp.b 20000000 10300000 × ；将程序烧写到Flash中，×是用16进制数表示的用户程序的大小，如ffff，表示用户程序小于64KB。

STEP3：复位系统，你的程序就会自动运行，这样，一个独立的嵌入式系统就开发成功了。

当然，绝大多数的用户需要在H9200E上运行嵌入式操作系统，如Linux等，以实现更为复杂的应用，其后的章节我们将描述将Linux操作系统内核下载的系统并进行应用开发的过程。

六、嵌入式 Linux 开发环境

“工欲善其事，必先利其器”，为提高嵌入式Linux的开发效率，一个完善的开发环境是必不可少的。在此我们描述建立一个完整、高效嵌入式Linux开发平台的基本方法以及常见的操作，以及如何通过该环境对H9200E进行Linux下载操作。

6.1 嵌入式 Linux 开发环境的基本结构

嵌入式 Linux 开发环境一般由如下几部分构成：Linux 服务器（宿主机）、工作站、嵌入式目标系统和将它们连接在一起的网络环境，其具体结构如图 6－1 所示。

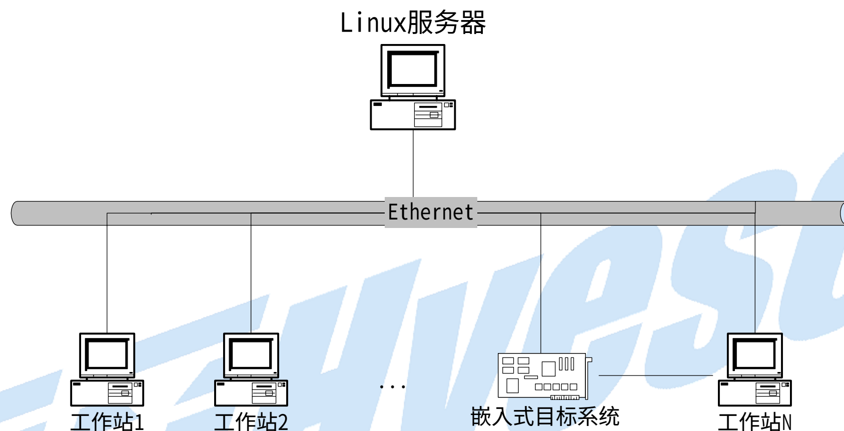


图 6－1　常见嵌入式 Linux 开发环境的结构

在图 6－1 所示的嵌入式 Linux 开发环境中，Linux 服务器作为嵌入式 Linux 内核编译、应用程序编译的公共平台，一般由单独的一台 PC 机充当，安装常用的桌面标准 Linux 操作系统，如 RedHat Linux 等。

工作站即为普通的局域网络计算机，可以是一台或多台，以支持小组项目开发，工作站一般安装常用的 Windows 操作系统，仍然可以完成各种日常工作，当需要使用 Linux 服务器资源时，可从工作站远程登录到 Linux 服务器，以完成各项需要的操作。

嵌入式目标系统是我们需要开发的最终产品，可以根据开发需要与工作站连接（通常是通过串行接口或 USB 接口），或连接到局域网络中（通过以太网接口）。

工作站作为开发人员的操作终端，可能需要经常登录 Linux 服务器使用相关资源，并在 Linux 服务器和工作站之间进行文件的传输，因此，一般需要在工作站上安装 FTP 客户端程序（如 Cuteftp 等）和 Telnet 客户端程序（如 SecureCRT 等），当 Linux 服务器的 FTP 服务和 Telnet 服务运行时，开发人员就可以方便的在工作站和 Linux 服务器之间进行文件传输，并可以通过 Telnet 方式登陆到 Linux 服务器，并对其进行相关操作。

为便于理解，我们约定以后的操作方式如下：开发人员在其中的一台工作站进行操作，通过远程登陆的方式操作 Linux 服务器，并使用 FTP 方式在 Linux 服务器和工作站之间进行文件传输，同时 H9200E 系统需要与网络连接，H9200E 系统的串行接口与该工作站的 RS232 接口连接，使用工作站上的超级终端作为嵌入式目标系统输入输出终端。

显然，我们首先需要在网络中的 Linux 服务器安装桌面标准 Linux 操作系统（我们约定使用 RedHat Linux 9），并需要保证局域网络畅通。

6.2 在 Linux 服务器上安装交叉编译工具

开发基于 Linux 的应用程序，一般的步骤是先编写程序源码，然后编译调试，最终生成可执行程序。

目前，已有各种基于 X86 架构的 GUN 开发工具集，如 C 编译器 GCC，C++编译器 G++、连接器、调试器等，这些工具一般都作为标准 Linux 的一部分存在。但使用这些工具编译出来的可执行程序只能在基于 X86 架构的 CPU 上运行，因为编译器最终编译出来的是基于 X86 架构的机器码。

为使编译出的可执行代码能在诸如 ARM 或其它体系架构的 CPU 上运行，必须有一个编译工具，这个工具应该能运行在基于 X86 架构的 Linux 服务器上，但通过它编译生成的可执行代码能够支持在其他体系结构的 CPU 上运行，这样的一个编译工具就叫交叉编译工具。

为什么要使用交叉编译呢？这是由于嵌入式设备没有足够的内存以及存储资源来完成其编译过程，所以人们想了个办法，就在主机上完成针对目标机的代码编译生成，这个过程称为交叉编译。这也目前大多数嵌入式设备开发所使用的编译方式。

我们如何得到支持特定微处理器架构的交叉编译工具呢？交叉编译工具一般由专门的机构负责维护，可以从他们的网站上免费取得，但我们已在 CD 中提供了基于 H9200E 开发的 Linux 交叉编译工具，以下我们描述交叉编译环境的建立过程：

我们首先以 Root 身份在 Linux 服务器上新建一个用于嵌入式 Linux 开发的工作目录：/home/work，以后所有的开发工作都在这个目录下进行。

编译工具以压缩包的形式，由我们提供，文件名为：cross-2.95.3.tar.bz2，包括arm-gcc编译器和一些实用程序，位于光盘的目录：Software\Linux\tools；

在Linux服务器的/usr/local目录新建子目录arm，从某工作站上通过FTP方式，将光盘中的文件cross-2.95.3.tar.bz2传输到Linux服务器的/usr/local/arm目录，然后以远程登录的方式，进行解压安装的操作：

```
#bunzip2 cross-2.95.3.tar.bz2
#tar xvf cross-2.95.3.tar
```

当以上的工作完成以后，就会在当前目录生成一个名为2.95.3的文件夹，嵌入式编译工具就安装在这个目录中，可以完成源代码的编译。

6.3 嵌入式 Linux 内核的配置与编译

Linux之所以能成为一种流行的嵌入式操作系统，除具有功能强大、高性能、稳定性好以及源代码开发等优势以外，其最大的特点是Linux内核具有非常良好的结构，可由用户根据特定的系统需求，对内核进行配置或裁减，而这一特点恰恰满足了嵌入式应用的差异性需求。

嵌入式系统最大的特征是差异性，几乎每一个嵌入式应用都是唯一的，这种差异性体现在硬件方面，就表现为系统可以采用不同的微处理器架构，如 X86、ARM、PowerPC、MIPS 等，同时，即使采用同一种微处理器架构，不同生产商生产的微处理器也会有不小的差异，运行在这些嵌入式微处理器上的操作系统也会有所相同，因此，如何得到一个适合在某个特定的嵌入式系统上运行的 Linux 内核，是我们首先要解决的问题。

我们提供可以稳定运行在H9200E的嵌入式Linux内核包，位于光盘目录：Software\linux\source。

将该 Linux 内核源代码包 linux-2.4.27-20xxxxxx-AT91RM9200.tar.gz 拷贝到 Linux 服务器上我们约定的工作目录：/home/work，然后执行如下命令解压：

```
#gunzip linux-2.4.27-20xxxxxx-AT91RM9200.tar.gz
```

该命令执行完毕后，生成解压文件linux-2.4.27-20xxxxxx-AT91RM9200.tar，然后再执行如下命令：

#tar xvf linux-2.4.27-20xxxxxx-AT91RM9200.tar

该命令执行完毕后，生成目录 linux-2.4.27-20xxxxxx-AT91RM9200。进入该目录，我们进行随后的操作。

注意文件名中的“ xxxxxx ”为文件生成的时间，我们会定期更新 Linux 内核的版本。

Linux 内核的配置（或裁剪）是与嵌入式系统的应用需求相适应的。尽管 Linux 内核功能强大，支持的设备众多，但对于一个特定的嵌入式应用来说，可能只会使用到其中的部分功能，而对于其它不使用的部分，如果让它驻留在系统中，不但耗费系统资源，同时还会增加系统安全隐患，因此，内核配置的目的，就是保留系统需要的功能，去掉不必要的部分，使操作系统以最精简、最优化的状态运行。

Linux 内核支持三种配置方式，第一种方式是基于命令行的问答方式，针对每一个内核配置选项会有一个提问，回答“y”包含该项功能，回答“n”则不包含该项功能。通过执行 make config 可以开始第一种方式。

第二种是菜单式的，用户可以在 Linux 服务器或网络中的某个工作站进行操作，执行 make menuconfig 命令以后，会出现一个配置菜单，通过该菜单可以很方便的进行内核的配置。

第三种方法也是采用菜单方式进行配置的，但必须在 Linux 服务器上执行。通过执行 make xconfig 可以开始第三种方式。

显然，三种方式的实质是相同的，我们通常使用第二种方式进行配置，这种方式简单明了，受条件制约小。

执行如下命令：

make menuconfig

注意：该命令可能需要超级用户权限。

系统出现图 6－2 所示的内核配置界面。



图 6－2　嵌入式 Linux 内核的配置界面

用户也许对配置选项的意义不是完全明白，可以先不用管他，我们已经对内核做了比较合理的配置，建议在开始接触时不要做修改。

当用户在根据自己的系统需求配置好内核，退出配置菜单时，需要保存修改后的内核配置，如图 6－3 所示。



图 6－3　嵌入式 Linux 内核配置保存界面

若用户选择不保存，则进行的所有配置操作无效，内核配置仍然为原来的状态，若选择保存，系统会在当前目录下生成一个.config 文件，其后要进行的内核编译就是根据这个.config 文件来进行条件编译以生成相应的可执行文件的。

当完成对 Linux 内核的配置以后，内核仍然以源代码的方式存在，不能直接下载到嵌入式系统运行，因此，需要对内核进行编译，生成最终可以在嵌入式系统上运行的可执行代码。

依次执行如下命令，对Linux内核进行编译：

```
#make clean          ；该命令用于清除旧的编译文件
#make dep            ；该命令用于生成系统相应的依赖文件
# ./automake
```

当你以极大的耐心执行完以上的步骤，如果运气不错的话，会在当前目录下生成内核压缩映象文件 uImage。

6.4　通过 UBoot 下载内核到 H9200E 运行

H9200E系统使用jffs2根文件系统启动方式，这样可以实现根文件系统的各种文件的掉电保存功能。以下我们描述将生成的Linux内核下载到系统并启动的过程。

H9200E在出厂时，已将Linux内核写入Flash，你可以直接上电运行，但如果你对Flash进行了擦除操作，或删除了系统的重要文件，你可以按如下的步骤将Linux内核写入H9200E并运行。

首先将CD中我们提供的包含内核映象文件及相关工具的文件夹：Software\linux\bin全部拷贝到某Windows工作站上，该文件夹包含如下3个文件：TFTPSRV、uImage和jffs2.img，我们在以下的操作中都需要用到。

在以上的3个文件中，TFTPSRV为一个运行在Windows平台上的TFTP服务器，通过运行在H9200E上的TFTP客户程序，可以完成在Windows工作站与H9200E的网络文件传输，uImage为Linux源码编译所生成的映象文件，关于jffs2.img文件的生成方法，用户可以参考附录D。

1、 设置运行Linux的环境变量

将H9200E的DEBUG_UART与Windows工作站的串口连接，然后启动Windows工作站上的超级终端，给H9200E上电，运行Uboot，执行如下命令设置Linux运行的环境变量：

```
Uboot>setenv bootargs root=/dev/mtdblock/3 console=ttyS0,115200 mem=32M
                                                  ；设置启动系统的环境变量
```

Uboot>setenv image cp.b 10020000 21000000 e0000
Uboot>setenv ramdisk run
Uboot>setenv boot bootm 21000000
Uboot>setenv bootcmd run image\;run boot
Uboot>setenv ethaddr 00:12:34:56:78:9a　　　；可能会出现“ Can't overwrite ethaddr ”的提示，忽略。
Uboot>setenv ipaddr 192.168.0.16　　　；设置H9200E的IP地址，用户可修改
Uboot>setenv serverip 192.168.0.30　　　；设置Windows工作站IP地址，用户可修改
Uboot>saveenv　　　；保存环境变量

到此，参数就保存到Flash里了，你可以用如下命令看看你的设置是否正确：

Uboot>printenv

注意此时有2个设备的IP地址：Windows工作站为192.168.0.30、H9200E为192.168.0.16，用户可以根据自身网络的实际情况对各自的IP地址进行修改。

提示：用户可以采用整行拷贝的方式将命令输入到超级终端，不用逐个输入。

2、从Windows工作站下载Linux内核到H9200E运行

将H9200E联入局域网络，然后运行Windows工作站的TFTPSRV应用程序并最小化，继续执行如下操作：

Uboot>tftp 21000000 uImage　　　；从Windows工作站下载文件到系统SDRAM
Uboot> cp.b 21000000 10020000 e0000　　　；将文件从SDRAM拷贝到系统Flash中

这个过程需要一定的时间，请耐心等待，不要断电，直到重新出现UBoot提示符。当出现UBoot提示符后，继续进行如下操作：

Uboot> tftp 21100000 jffs2.img
Uboot> cp.b 21100000 10100000 260000

这个过程需要约10分钟时间，请耐心等待，不要断电，直到重新出现UBoot提示符。

当以上的操作完成以后，请重新复位系统，此时H9200E就可以作为一个独立的嵌入式系统启动了，当Linux启动以后，H9200E的4MB Nor Flash分为三个MTD分区，分区号和大小如下：

/dev/mtd/1	“bootloader”	0x20000
/dev/mtd/2	“uImage”	0xe0000
/dev/mtd/3	“jffs2root”	0x100000

H9200E系统的64MB Nand Flash也分为三个MTD分区，并通过yaffs系统将3个mtd分区挂载到不同目录上，其分区号、大小和挂载目录分别如下：

/dev/mtd/4	16M	/mnt/Nand1
/dev/mtd/5	16M	/mnt/Nand2
/dev/mtd/6	32M	/mnt/Nand3

系统启动后，根目录下的/tmp和/var子目录挂载在系统SDRAM上，用户在系统开发过程中形成的各种不需要掉电保存的临时文件就可以暂时存放到这两个目录中。

H9200E系统第一次启动后，网卡会自动生成一个随机的MAC地址，并将其存入/usr/etc/Mac.txt文件，

以后每次启动时，将直接从该文件中读取第一次配置的MAC地址进行配置。同时，系统的网络IP地址被默认配置为192.168.0.12，启动后可根据网络环境使用ifconfig命令进行修改，当然，用户若想让各项配置命令在系统启动后自动生效，可修改/usr/etc/rc.local文件中关于IP地址的设定。

到此，你的嵌入式Linux系统已经启动了，你可以试着运行其中的各种程序。

其后的章节，我们将继续描述基于Linux的应用程序开发及设备驱动程序的开发。



七、如何在 H9200E 上开发 Linux 应用程序

到目前为止，我们已经组建了基于AT91RM200的硬件系统和嵌入式操作系统Linux的平台，而最终完成具体任务是由应用程序来承担的，如前所述，ARM嵌入式微处理器最诱人的一个特点就是能够流畅的运行各种嵌入式操作系统，有了嵌入式操作系统，你就可以像在PC平台上一样，实现各种功能强大的应用。

本节的内容，就是描述如何在H9200E的Linux操作系统上，完成一个最简单的应用程序。

7.1 Linux 应用程序的开发流程

基于嵌入式 Linux 的应用程序开发流程，与基于 Windows 的应用程序开发有很大的不同。在 Windows 环境中，开发者习惯使用各种集成编译开发环境，完成程序编辑、编译和运行。

而在 Linux 下开发应用程序，目前还缺乏比较简单、高效的开发工具和手段，同时，由于应用程序的最终运行平台是嵌入式目标系统，但程序开发仍然需要在 PC＋Linux 平台完成，因此，在程序的开发与调试过程中，需要频繁的在 Linux 服务器和嵌入式目标平台交换信息。由于以上这些原因，基于嵌入式 Linux 的应用程序开发，还是一个相对比较复杂的过程，但随着嵌入式 Linux 的推广使用，基于嵌入式 Linux 应用程序的开发会变得越来越方便和容易。

有几种开发模式可以用于基于嵌入式 Linux 应用程序的开发，但最常用的方式有如下两种，即 FTP 方式和 NFS 方式。

FTP 方式：

FTP 方式是首先在 Linux 服务器上编辑源文件，然后交叉编译，最后生成可执行文件，并将生成的可执行文件通过 FTP 方式下载到嵌入式目标系统上运行。

因此，当开发者在 Linux 服务器上完成了应用程序的编辑和编译，生成可执行文件以后，需要在嵌入式目标系统上通过 FTP 方式，下载编译好的可执行文件到嵌入式目标系统上运行，如果程序运行错误，则回到 Linux 服务器上修改源文件并重新编译、下载运行，直到程序正确运行为止。

FTP 方式需要 Linux 服务器运行 FTP 服务，同时要求嵌入式目标系统端运行 FTP 客户程序。

FTP 方式是一种常用的嵌入式 Linux 应用程序开发方式，其流程如图 7－1 所示。

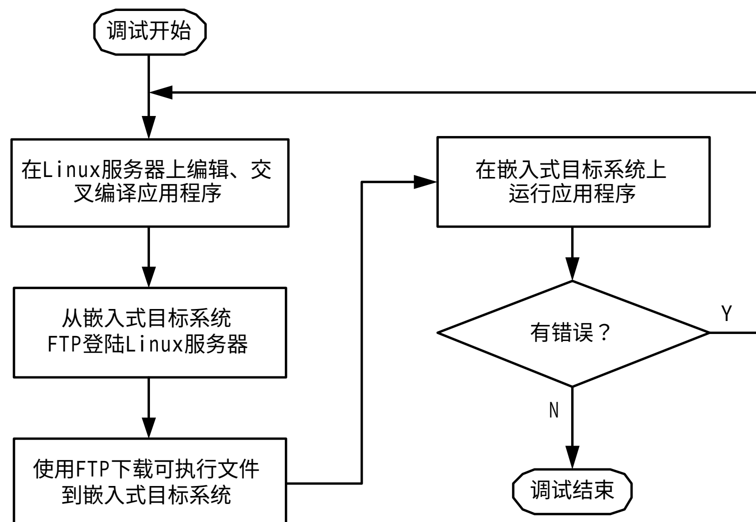


图 7－1　FTP 方式的应用程序开发流程

NFS 方式：

NFS 方式也是首先在 Linux 服务器上编辑源文件，然后交叉编译，最后生成可执行文件，但生成的可执行文件不再通过 FTP 等方式下载到嵌入式目标系统，而是在嵌入式目标系统端通过 NFS 方式挂载 Linux 服务器的共享分区，让应用程序直接运行在嵌入式目标系统，并进行调试。

NFS 方式需要 Linux 服务器端和嵌入式目标系统端均能支持 NFS 文件系统。
与其他方式相比较，NFS 方式具有较高的调试效率，是一种经常采用的方法。NFS 方式的开发流程如图 7－2 所示。

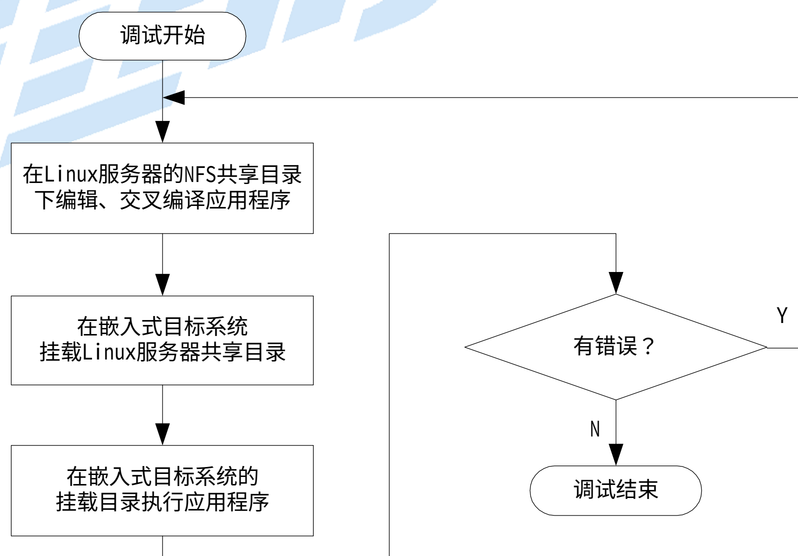


图 7－2　NFS 方式的应用程序开发流程

当开发人员完成了应用程序的调试与开发，可以将调试好的应用程序下载到嵌入式目标系统的 Flash 文件系统，或直接编译到嵌入式 Linux 内核并烧写到系统的 Flash，从而最终形成一个独立的嵌入式应用系

统。

我们推荐用户使用 NFS 方式进行应用程序的方法，关于 NFS 环境的建立方法详见附录 C。

在以下的部分，我们将以一个简单的示例程序来说明基于嵌入式 Linux 应用程序的开发过程。

7.2 一个简单的应用程序

我们使用一个最简单的示例，来说明基于嵌入式 Linux 的应用程序开发过程。示例程序完成 1～100 的累加求和，运行时输出每一步的累加结果和最后的总和。

FTP 方式：

第一步：编辑应用程序。

在 Linux 服务器的工作目录/home/work，使用 VI 文本编辑器编辑应用程序，应用程序名为 test.c：

```
# vi test.c
```

程序代码如下：

```
/* this is a simple calculate program */
#include <stdio.h>
int main(void)
{
	int i,sum=0,temp_sum=0;
	for(i=1;i<=10;i++)
		{
		sum=sum+i;
		temp_sum=sum;
		printf("temp_sum = %d\n",temp_sum);
		}
	printf("Calculate finished! sum = %d\n",sum);
	return 0;
}
```

编辑完源代码后，保存文件并退出 VI。

第二步：编译应用程序。

键入如下命令编译应用程序：

```
# arm-linux-gcc -o test test.c
```

当命令执行完毕以后，会在当前目录下生成可执行文件 test。

第三步：通过 FTP 方式将可执行文件 test 下载到嵌入式目标系统运行。

启动嵌入式目标系统的 Linux，使用超级终端作为其用户操作界面，切换到 Linux 的可写目录，如\var，键入如下命令使用 FTP 方式登陆 Linux 服务器：

```
# ftp 192.168.0.20
```

192.168.0.20 为 Linux 服务器的 IP 地址。输入合法的用户名和密码，登陆到 Linux 服务器，执行如下命令将可执行文件 test 下载到嵌入式目标系统的当前目录。

```
# bin                ；以二进制方式传输文件
# get test           ；将文件 test 下载到 H9200E 的当前目录
# bye
```

当以上命令执行完毕以后，可执行文件 test 下载到嵌入式目标系统的当前目录。

第四步：修改文件属性，运行应用程序。

可执行文件 test 下载到嵌入式目标系统后，可能不具有可执行属性，键入如下命令修改文件属性，并运行应用程序：

```
# chmod 777 test     ；修改文件属性为可执行；
# ./test             ；运行应用程序；
```

此时，用户可以看到应用程序的运行效果，输出每一步的求和结果和总的累加和。

NFS 方式：

第一步：编辑应用程序。

在 Linux 服务器的工作目录/home/work，使用 VI 文本编辑器编辑应用程序，应用程序名为 test.c。

注意该目录必须是 Linux 服务器的输出共享目录。

```
# vi test.c
```

程序代码如下：

```
/* this is a simple calculate program */
#include <stdio.h>
int main(void)
{
    int i,sum=0,temp_sum=0;
    for(i=1;i<=10;i++)
            {
            sum=sum+i;
            temp_sum=sum;
            printf("temp_sum = %d\n",temp_sum);
            }
    printf("Calculate finished! sum = %d\n",sum);
    return 0;
}
```

编辑完源代码后，保存文件并退出 VI。

第二步：编译应用程序。

键入如下命令编译应用程序：

```
# arm-linux-gcc -o test test.c
```

当命令执行完毕以后，会在当前目录下生成可执行文件 test。

第三步：在嵌入式目标系统端挂载 Linux 服务器的输出共享目录，并运行应用程序。

启动嵌入式目标系统的 Linux，使用超级终端作为其用户操作界面，键入如下命令挂载 Linux 服务器的输出共享目录：

mkdir /mnt/nfs ；建立 Linux 服务器输出共享目录的挂载点；

mount －t nfs 192.168.0.20:/home/work /mnt/nfs -o nolock

cd /mnt/nfs

ls

此时，嵌入式目标系统端所显示的内容即为 Linux 服务器的输出目录的内容，可执行文件 test 就在该共享目录下，运行该程序：

./test

此时，用户可以看到应用程序的运行效果，输出每一步的求和结果和总的累加和。

显然，由于 NFS 方式省去可执行文件从 Linux 服务器传输到嵌入式目标系统端的过程，当开发的应用程序比较复杂时，采用该方式可以大大提高应用程序的开发效率。

7.3 固化应用程序并自动运行

无论是使用FTP方式，还是NFS，应用程序都没有固化到系统中，但在实际的嵌入式应用中，用户需要将已经调试完成的程序固化到系统的Flash存储器，并希望系统启动时可以自动运行。

应用程序可以以静态的方式，放入Linux源代码中，然后重新编译Linux源码并下载到嵌入式系统运行，但这种方式很不灵活，只要用户修改了应用程序，就需要重新编译并下载，使用非常不方便。

H9200E在Nor Flash和NAND Flash上均实现了文件系统支持，用户可以象操作PC的硬盘一样，对相关目录进行读写，这些目录都具有掉电不丢失数据的特性。

在H9200E系统Linux启动以后，用户可以使用命令df，查看系统的分区信息：

df

此时显示系统的分区信息：

/dev/mtd/4	16M	/mnt/Nand1
/dev/mtd/5	16M	/mnt/Nand2
/dev/mtd/6	32M	/mnt/Nand3

用户可以将调试好的应用程序，如前所述的test，通过NFS或FTP方式拷贝到某个目录下，只要用户不将其删除，该文件就会一直存在。

然后用户可以编辑文件/usr/etc/rc.local，该文件为系统启动时的自动执行文件，用户可以将需要启动运行的程序路径及文件名添加到这里，就可以实现在系统启动时的程序自动运行。

八、如何在 H9200E 上开发 Linux 设备驱动程序

Linux最诱人和最有挑战性的部分就是设备驱动程序的编写，同时也是一个不太好掌握的难点，当你通过学习、实践，能够随心所欲的编写设备驱动程序时，你就距离嵌入式系统开发的高手不远了。

有许多专业的书籍讲解Linux的设备驱动程序的开发，当然，如果有时间，看看这些书是非常有好处的，至少你可以明白关于设备驱动程序的一些基本概念：什么是设备驱动程序？为什么要编写设备驱动程序？设备驱动程序是如何工作的？等等。

但在此我们不对这些问题进行详细的描述，否则就有点喧宾夺主了。

其实，Linux内核已经为我们提供了大量的设备驱动程序，如果我们的设备是标准设备，比如usb设备，一般就不需要你自己从头开始写驱动，仅仅需要修改Linux自带的驱动程序，或者可以找到现成的驱动程序作为补丁打上去。

但在有些情况下，例如你自己设计了一些非标准的设备，比如ADC、DAC、用户自定义键盘等，要在Linux下使用这些设备，就要求你自己动手来写这些设备的驱动程序了。

Linux的设备驱动程序根据设备的类型和特点具有不同的复杂度，表面看起来千差万别，而实际上这些驱动却具有一些共性，只要掌握了这些驱动程序的“ 套路 ”，再加上对设备的熟悉程度就可以比较轻松的开发自己的驱动程序了。

下面我们结合简单的例子来说明如何在标准Linux下开发设备驱动程序。

8.1 编写设备驱动程序

我们使用一个比较简单的例程，来说明设备驱动程序的基本编写方法，该设备驱动程序完成AT91RM9200的部分GPIO驱动。

该示例程序的源码位于CD：\software\examples\GPIO\leddrv，文件名为at91_led.c，设备驱动程序的编写与编译均采用模块方式，与静态加入内核的方式不同，采用动态加载模块的方式可以免去反复编译内核的麻烦，大大提高开发效率。

1、设备驱动程序的基本结构

GPIO 驱动程序的主要结构描述如下：

```
// linux/drivers/at91/led/at91_led.c
// Copyright (C) Hyesco Technology Co.,Ltd
// History:     Original version

#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/mm.h>
```

```
#include <linux/poll.h>
#include <linux/slab.h>
#include <linux/ioport.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/fcntl.h>
#include <linux/rtc.h>
#include <linux/miscdevice.h>
#include <linux/string.h>
#include <linux/proc_fs.h>
#include <asm/bitops.h>

#include <asm/arch/AT91RM9200_SYS.h>
#include "ledcommon.h"

#define LED_MODULE_NAME "AT91_LED"

#define LED0_DATA(x)
    if (0==(x)) AT91_SYS->PIOB_CODR|=(0x01<<13);
    else AT91_SYS->PIOB_SODR|=(0x01<<13);          //对应于AT91RM9200的PB13
#define LED1_DATA(x)
    if (0==(x))   AT91_SYS->PIOB_CODR|=(0x01<<14);
    else AT91_SYS->PIOB_SODR|=(0x01<<14);          //对应于AT91RM9200的PB14
#define LED2_DATA(x)
    if (0==(x))   AT91_SYS->PIOB_CODR|=(0x01<<15);
    else AT91_SYS->PIOB_SODR|=(0x01<<15);          //对应于AT91RM9200的PB15
#define LED3_DATA(x)
    if (0==(x))   AT91_SYS->PIOB_CODR|=(0x01<<16);
    else AT91_SYS->PIOB_SODR|=(0x01<<16);          //对应于AT91RM9200的PB16

#define LEDPIO 0x0001e000

//LED Status
unsigned char gLEDStatus;

static int   LED_open(struct inode *inode, struct file *file)
{
    MOD_INC_USE_COUNT;
    return 0;
}

static int   LED_release(struct inode *inode, struct file *file)
```

```
{
    MOD_DEC_USE_COUNT;
    return 0;
}

static ssize_t LED_write(struct file * filp, const char * buf, size_t count, loff_t * l)
{
    LEDStatus status;
    if (sizeof(LEDStatus)!=count)
    {
        printk("<1>LED_write data is not LEDStatus\n");
        return -1;
    }
    if (copy_from_user(&status,buf, count))
        return -EFAULT;

    gLEDStatus = gLEDStatus & ~(0x01<<(status.LEDNum-1))
|(status.LEDValue<<(status.LEDNum-1));
    switch (status.LEDNum)
    {
    case 1:
        LED0_DATA(gLEDStatus & 0x01)
        break;
    case 2:
        LED1_DATA((gLEDStatus & 0x02) >> 1)
        break;
    case 3:
        LED2_DATA((gLEDStatus & 0x04) >> 2)
        break;
    case 4:
        LED3_DATA((gLEDStatus & 0x08) >> 3)
        break;
    default:
    }
    return count;
}

static ssize_t LED_read(struct file * filp, char * buf, size_t count, loff_t * l)
{
    LEDStatus status;
    if (sizeof(LEDStatus)!=count)
    {
```

```
            printk("<1>LED_read data is not LEDStatus\n");
            return -1;
        }
        if (copy_from_user(&status,buf, count))
            return -EFAULT;

        status.LEDValue=(gLEDStatus & (0x01<<(status.LEDNum-1) ) )>> (status.LEDNum-1);
        return copy_to_user((void *)buf, &status, count);
}

static int LED_ioctl(struct inode *inode, struct file *file, uint cmd, ulong arg)
{
        LEDStatus status;
        switch (cmd)
        {
        case LED_ALLON:
            gLEDStatus = 0xff;
            AT91_SYS->PIOB_SODR|=LEDPIO;
            break;
        case LED_ALLOFF:
            gLEDStatus = 0;
            AT91_SYS->PIOB_CODR |=LEDPIO;
            break;
        default:
            printk("<1>LED_ioctl command error\n");
        }
        return 0;
}

static struct file_operations LED_fops=
{
        owner:          THIS_MODULE,
        open:           LED_open,
        release:        LED_release,
        write:          LED_write,
        read:           LED_read,
        ioctl:          LED_ioctl,
};

static int gMajor=251;
static int __init LED_at91_init(void)
{
```

```
        int result = 0;
        // PB13~PB16
        AT91_SYS->PIOB_PER |=LEDPIO;
        // enable output
        AT91_SYS->PIOB_OER |=LEDPIO;
        AT91_SYS->PIOB_OWER |=LEDPIO;
        // all off
        AT91_SYS->PIOB_CODR |=LEDPIO;
        gLEDStatus = 0;

        result= devfs_register_chrdev(gMajor, LED_MODULE_NAME, &LED_fops);
        if (result < 0) {
            printk("<1>Init_LED devfs_register_chrdev can't get major number 251\n");
            return result;
        }

        return 0;
}

static void __exit LED_at91_cleanup(void)
{
            devfs_unregister_chrdev(gMajor, LED_MODULE_NAME);
}

module_init(LED_at91_init);
module_exit(LED_at91_cleanup);

MODULE_AUTHOR("www.hyesco.com>");
MODULE_DESCRIPTION("AT91 LED Driver (AT91_LED)");
MODULE_LICENSE("GPL");
```

需要注意的是在标准Linux下，不允许对设备的物理地址直接访问，而必须转换为虚拟地址后才可访问。

在Linux的源码目录include/asm-arm/arch-at91rm9200/hardware.h中定义了AT91_IO_P2V宏来完成物理地址到虚拟地址的映射，再定义具体外设虚拟地址。在本例中，通过：

#include <asm/arch/AT91RM9200_SYS.h>

就可以在函数中定义变量AT91PS_SYS sys = (AT91PS_SYS) AT91C_VA_BASE_SYS，然后通过sys结构来访问AT91PS_SYS中的成员了。

注册模块：

在Linux中，任何设备都有一个主设备号作为操作系统区别不同驱动程序的标识。在本例中为LED设备指定了一个设备号为251：

static int gMajor=251;
然后使用注册函数向系统注册该设备：
result= devfs_register_chrdev(gMajor, LED_MODULE_NAME, &LED_fops);

注销模块：

注销模块是和注册模块相对应的操作，当卸载模块的时候，系统将注销模块对应的设备号。
devfs_unregister_chrdev(gMajor, LED_MODULE_NAME);

编译驱动程序：

我们编译如下Makefile文件，对模块进行编译：

```
CC = /usr/local/arm/2.95.3/bin/arm-linux-gcc
KERNELDIR = /home/wjh/linux-2.4.27-vrs1-ATMEL
CFLAGS = -DMODULE -I$(KERNELDIR)/include -Wall -Wstrict-prototypes -Wno-trigraphs -Os
-fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LINUX_ARM_ARCH__=4
-march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm    -nostdinc -I
/usr/local/arm/2.95.3/lib/gcc-lib/arm-linux/2.95.3/include

at91_led.o: at91_led.c
	$(CC) $(CFLAGS) -c $^

.PHONY: clean
clean:
	-rm -f *.o

distclean:
	@make clean
	rm -f tags *~
```

在当前目录下执行命令make，对程序进行编译：

```
# make
```

如果程序不能通过编译，回到第一步修改源文件，当编译正确完成以后，会在当前目录下生成文件名为at91_led.o的可加载模块。

模块加载：

用户可以通过NFS或FTP方式，将生成的可加载模块文件拷贝到H9200E的/lib/modules目录，然后执行如下命令将模块加载到Linux内核中，并建立字符设备文件LED：

```
# insmod /lib/modules/at91_led.o > /dev/null
```

mknod /dev/LED c 251 1

当完成以上操作以后，用户就可以在/dev目录下看到新加入的设备文件LED。如果用户希望系统启动时，能自动加载模块并建立设备文件，可以将以上命令加入/usr/etc/rc.local文件的适当位置即可。

8.2 编写应用程序访问设备

从Linux操作系统的工作原理上看，设备驱动程序是被动的，即设备驱动程序的作用要通过应用程序来实现，因此，在一个具体的应用中，完成设备驱动程序仅仅是第一步，用户需要编写应用程序对设备程序进行调用，从而实现特定的功能。

编写测试程序

我们编写如下示例应用程序测试LED设备。

该示例程序的源码位于CD：\software\examples\GPIO\testLED，文件名为testLED.c。

```
//	Description：	test led driver
//
//	Copyright (C)	Hyesco Technology Co.,Ltd
//

#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include "ledcommon.h"

int main(int argc,char **argv)
{
	int fd,i,retval;
	LEDStatus status;

	fd = open("/dev/LED",O_RDWR);			；打开设备文件
	if (fd<0)
	{
		printf("open device LED error!\n");
		return 0;
	}

	while(1)
```

```
{
    //read and write
    for (i=1;i<9;i++)
    {
        status.LEDNum=i;
        retval=read(fd, &status, sizeof(LEDStatus));
        if (retval<0)
            printf("Read Error\n");
        printf("LED %d=%d\n",i,status.LEDValue);

        status.LEDValue=1;
        retval=write(fd, &status, sizeof(LEDStatus));
        if (retval<0)
            printf("Write Error\n");

        retval=read(fd, &status, sizeof(LEDStatus));
        if (retval<0)
            printf("Read Error\n");
        printf("LED %d=%d\n",i,status.LEDValue);

        sleep(1);
    }

    for (i=1;i<9;i++)
    {
        status.LEDNum=i;
        status.LEDValue=0;
        retval=write(fd, &status, sizeof(LEDStatus));
        if (retval<0)
            printf("Write Error\n");

        sleep(1);
    }
    //all on
    retval=ioctl(fd, LED_ALLON, 0);
    if (retval<0)
        printf("ioctl Error\n");
    sleep(1);

    // all off
    retval=ioctl(fd, LED_ALLOFF, 0);
    if (retval<0)
```

```
                printf("ioctl Error\n");
            sleep(1);
            }

        close(fd);
        return 0;
    }
```

该程序实现对 GPIO 的高低电平设置，以及一些其他的操作。

编译测试程序

我们编译如下Makefile文件，对测试程序进行编译：

```
testLED: testLED.c
    /usr/local/arm/2.95.3/bin/arm-linux-gcc -o testLED    testLED.c
clean:
    rm -f testLED
```

在当前目录下执行命令make，对程序进行编译：

```
# make
```

如果程序不能通过编译，回到第一步修改源文件，当编译正确完成以后，会在当前目录下生成文件名为testLED的可执行文件。

8.3 测试设备

用户可以通过NFS或FTP方式，将生成的测试程序文件拷贝到H9200E的某个目录，然后执行如下命令运行该文件：

```
#./testLED
```

当用户在对应 GPIO 连接 LED 显示器时，可以看到相应的效果。

用户也可以将以上命令加入/usr/etc/rc.local 文件的适当位置即可实现程序在系统启动时的自动运行。

你瞧，其实很简单。

8.4 其他的设备文件例程

为方便用户使用，我们还提供了系统的其他一些设备驱动程序和测试程序，用户可以作为系统开发的参考。

用户可以在 CD：\software\examples\下找到这些设备驱动程序和测试例程的源码，以及使用方法的描

述。

我们的用户手册到此就要结束了，如果你还有意犹未尽的感觉，祝贺你，你会在不久的将来成为一个嵌入式开发的行家。

如果你还需要支持和帮助，请随时与我们联系，谢谢读完如此冗长的文档！



附录 A Minicom 的设置

minicom 是 linux 下的一个串口通讯程序。它类似于 windows 下的超级终端。可以利用它来和目标实验板进行通讯。

1、 输入‘ minicom -s ’

```
[configuration]
Filenames and paths
File transfer protocols
Serial port setup
Modem and dialing
Screen and keyboard
Save setup as dfl
Save setup as..
Exit
```

2、 将光标移到‘ serial port setup ’并且回车。出现如下画面：

```
A -    Serial Device      : /dev/modem
B - Lockfile Location     : /var/lock
C -   Callin Program      :
D -  Callout Program      :
E -    Bps/Par/Bits       :115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No

   Change which setting?
```

3、 输入‘ E ’来改变通讯参数。出现如下画面：

```
[Comm Parameters]

Current:115200 8N1

  Speed              Parity          Data

A: 300               L: None         S: 5
B: 1200              M: Even         T: 6
C: 2400              N: Odd          U: 7
D: 4800              O: Mark         V: 8
E: 9600              P: Space
F: 19200
G: 38400
H: 57600
I: 115200            Q: 8-N-1
J: 230400            R: 7-E-1

Choice, or <Enter> to exit?
```

4、别输入‘F’、‘Q’来选择 115200 的波特率和 8n1　然后按回车。返回上一级菜单。

5、键入‘F’将硬件控制设置为否。

6、键入“A”将 Serial Device 改为/dev/ttyS0(注意,‘S’为大写,如果实际联接的是 COM2 则应改为 ttyS1)设置后的画面如下所示：

```
A -     Serial Device        : /dev/ttyS0
B - Lockfile Location        : /var/lock
C -     Callin Program       :
D -   Callout Program        :
E -     Bps/Par/Bits         : 19200  8N1
F - Hardware Flow Control    : No
G - Software Flow Control    : No

    Change which setting?
```

7、回车接受设置。

8、返回主菜单后，将光标移到 Modem and dialing 处，回车。

9、将 Init string ………后面的字符删除。

10、　　将 Reset string………后面的字符删除

11、　　回车接受设置.返回主菜单。

12、　　光标移到‘save setup as dfl’并回车确认。

13、 将光标移到‘exit’回车确认完成设置。

14、 以后只要键入 minicom 后，就可以按上述设置的参数进行连接；而不用每次都进行设置。

minicom 所有的操作都以ctrl+A 开始，例如：退出为ctrl+A，松手后再按下Q，则弹出一个小框，选Yes即可退出minicom。



附录 B　　Uboot 的编译方法

我们提供给用户Uboot的源代码，共包含三个压缩格式的文件，位于光盘：Software\uboot\source目录下，当用户需要对Uboot进行修改和编译时，可以按照如下的操作步骤进行：

1、编译LOADER

将文件AT91RM9200-Loader.tar.gz拷贝到Linux服务器的某个目录，执行如下命令解压文件：

```
#tar xzvf AT91RM9200-Loader.tar.gz
```

当文件解压完成以后，会自动生成AT91RM9200-Loader目录，进入该目录进行编译，命令如下：

```
#cd AT91RM9200-Loader
#make clean
#make
```

执行完上述命令以后，就会在当前目录下生成新的loader.bin文件。

2、编译BOOT

将文件AT91RM9200-Boot.tar.gz拷贝到Linux服务器的某个目录，执行如下命令解压：

```
#tar xzvf AT91RM9200-Boot.tar.gz
```

当文件解压完成以后，会自动生成AT91RM9200-Boot目录，进入该目录进行编译，命令如下：

```
#cd AT91RM9200-Boot
#make clean
#make
```

执行完上述命令以后，就会在当前目录下生成新的boot.bin文件。

3、编译UBOOT

将文件u-boot-1.0.0.tar.gz拷贝到Linux服务器的某个目录，执行如下命令解压：

```
#tar xzvf u-boot-1.0.0.tar.gz
```

当文件解压完成以后，会自动生成u-boot-1.0.0目录，进入该目录进行编译，命令如下：

```
#cd u-boot-1.0.0
#make clean
#make all
```

执行完上述命令以后，就会在当前目录下生成新的uboot.bin文件，执行如下命令对uboot.bin进行压缩：

```
#gzip -c u-boot.bin>u-boot.gz
```

注意：以上的操作可能需要超级用户权限。

压缩完成以后，在当前目录下生成u-boot.gz，然后按照前文所描述过的步骤，将生成的各个文件烧写到Flash里就可以了。

附录 C NFS（网络文件系统）建立与配置方法

网络文件系统（NFS，Network File System）是一种将远程主机上的分区（目录）经网络挂载到本地系统的一种机制，通过对网络文件系统的支持，用户可以在本地系统上像操作本地分区一样来对远程主机的共享分区（目录）进行操作。

在嵌入式 Linux 的开发过程中，开发者需要在 Linux 服务器上进行所有的软件开发，交叉编译后，通用 FTP 方式将可执行文件下载到嵌入式系统运行，但这种方式不但效率低下，且无法实现在线的调试。

因此，可以通过建立 NFS，把 Linux 服务器上的特定分区共享到待调试的嵌入式目标系统上，就可以直接在嵌入式目标系统上操作 Linux 服务器，同时可以在线对程序进行调试和修改，大大的方便了软件的开发。因此，NFS 的是嵌入式 Linux 开发的一个重要的组成部分，本部分内容将详细说明如何配置嵌入式 Linux 的 NFS 开发环境。

嵌入式 Linux 的 NFS 开发环境的实现包括两个方面：一是 Linux 服务器端的 NFS 服务器支持；二是嵌入式目标系统的 NFS 客户端的支持。因此，NFS 开发环境的建立需要配置 Linux 服务器端和嵌入式目标系统端。

Linux 服务器端 NFS 服务器的配置

以 root 身份登陆 Linux 服务器，编辑/etc 目录下的共享目录配置文件 exports，指定共享目录及权限等。

执行如下命令编辑文件/etc/exports：

```
# vi /etc/exports
```

在该文件里添加如下内容：

```
/home/work        192.168.0.*（rw,sync,no_root_squash）
```

然后保存退出。

添加的内容表示：允许 ip 地址范围在 192.168.0.*的计算机以读写的权限来访问/home/work 目录。/home/work 也称为服务器输出共享目录。

括号内的参数意义描述如下：

rw：读/写权限，只读权限的参数为 ro；

sync：数据同步写入内存和硬盘，也可以使用 async，此时数据会先暂存于内存中，而不立即写入硬盘。

no_root_squash：NFS 服务器共享目录用户的属性，如果用户是 root，那么对于这个共享目录来说就具有 root 的权限。

接着执行如下命令，启动端口映射：

```
# /etc/rc.d/init.d/portmap start
```

最后执行如下命令启动 NFS 服务，此时 NFS 会激活守护进程，然后就开始监听 Client 端的请求：

```
# /etc/rc.d/init.d/nfs start
```

用户也可以重新启动 Linux 服务器，自动启动 NFS 服务。

在 NFS 服务器启动后，还需要检查 Linux 服务器的防火墙等设置（一般需要关闭防火墙服务），确保没有屏蔽掉 NFS 使用的端口和允许通信的主机，主要是检查 Linux 服务器 iptables，ipchains 等选项的设

置，以及/etc/hosts.deny，/etc/hosts.allow 文件。

我们首先在 Linux 服务器上进行 NFS 服务器的回环测试，验证共享目录是否能够被访问。在 Linux 服务器上运行如下命令：

```
# mount –t nfs 192.168.0.20:/home/work /mnt
# ls /mnt
```

命令将 Linux 服务器的 NFS 输出共享目录挂载到/mnt 目录下，因此，如果 NFS 正常工作，应该能够在/mnt 目录看到/home/work 共享目录中的内容。

注意：我们 Linux 服务器 IP 地址是 192.168.0.20，用户如有不同，请对命令中的 IP 地址进行相应的修改。

嵌入式目标系统 NFS 客户端的配置

在 Linux 服务器设置好后，还需要对客户端进行相关配置。

我们已在 H9200E 系统所提供给用户的 Linux 内核进行了相应的配置，用户可以省略这一步的相关操作。

在嵌入式目标系统的 Linux Shell 下，执行如下命令来进行 NFS 共享目录挂载：

```
# mkdir /mnt/nfs                //建立 Linux 服务器输出共享目录的挂载点；
# mount –t nfs 192.168.0.20:/home/work /mnt/nfs –o nolock
# cd /mnt/nfs
# ls
```

此时，嵌入式目标系统端所显示的内容即为 Linux 服务器的输出目录的内容，即 Linux 服务器的输出目录/home/work 通过 NFS 映射到了嵌入式目标系统的/mnt/nfs 目录。

用户可以用增/删/修改文件的方式来验证实际效果。

mount 命令中的 192.168.0.20 为 Linux 服务器的 IP 地址，/home/work 为 Linux 服务器端所配置的共享输出目录，/mnt/nfs 为嵌入式设备上的本地目录。

附录 D　Busybox 及编译与使用方法

什么是Busybox呢？

从名字上直接理解，Busybox是一个“繁忙的盒子”，其实，Busybox可以理解为一个Linux的命令集合，我们在进行Linux操作时所需要的常用命令，都可以在Busybox里找到，但Busybox又不是简单的将所有的命令集合在一起，它采用了一种非常巧妙的方式，即“使用一个程序完成所有的事”。

平时我们用ls、vi等命令，都要用到glibc的相关调用，所以如果每个命令都静态链接这些调用，每个命令都会很大，因此在通常的发行版中，都会动态链接glibc，可是glibc的动态库本身就很大，这在PC＋Linux平时上还可以接受，但在嵌入式系统中，这就太大了，而且又不是所有的库函数都使用。

一般采用两种解决办法，一种是裁剪glibc，另一种就是Busybox的办法，即把ls、vi等程序的main函数改一下名，全部链接在一起，然后静态链接glibc，这样，只有需要的调用才会链接进来，整个Busybox程序可能都比glibc的动态库小。

因此，Busybox的工作原理是根据文件名来决定用户想调用的是那个程序，例如，如果你的busybox程序的文件名是ls，运行的就是ls，是vi，就运行vi。

我们在H9200E系统中提供了Busybox1.00源码，用户可以在使用的过程中，添加一些常用的命令到系统中。源码位于光盘：\Software\linux\busybox。

其编译与使用步骤描述如下：

1、编译Busybox

将文件busybox-1.00.tar.gz拷贝到Linux服务器的/home/work目录，执行如下命令解压文件：

#tar xzvf AT91RM9200-busybox-1.00.tar.gz

当文件解压完成以后，会自动生成busybox-1.00目录，进入该目录，并执行如下命令进行编译：

#make clean　　　　　　　　；清除旧的编译文件

#make menuconfig

执行该命令以后，进入配置主菜单。

首先进入菜单Build Options，选择“Do you want to build BusyBox with a Cross Compiler?”选项，并在“Cross Compiler prefix”栏中输入交叉编译器安装的位置，如/usr/local/arm/2.95.3/bin/arm-linux-，注意最后是arm-linux-，不能加其他字符或空格。

然后在“Any extra CFLAGS options for the compiler”栏中输入需要匹配的arm-linux系统源码目录中头文件所在的位置，如“/home/works/linux-2.4.27-XXXXXXX-ATMEL/include”，然后退出返回主菜单。

再根据系统需要，进入所需命令工具所在的子菜单中，选择相应的命令项。如需要“insmod”命令，就在“Linux Module Utilities”子菜单中选中。

选择完成所需的全部命令后，保存配置文件退出，并执行如下命令进行编译：

#make

#make install

当命令执行完毕以后，会在当前目录中会生成“_install”目录，进入该目录，可以看到其中有四个子目录：bin、linuxrc、sbin和usr。

2、将编译结果加入ramdisk

在/home/work目录下新建一个临时目录tmp，执行如下命令，将原有的ramdisk.gz解压并挂载到该目录下：

```
# gunzip ramdisk.gz
# mkdir tmp
# mount -o loop ramdisk tmp
```

然后执行如下命令，将_isntall目录中的文件拷贝到tmp目录，覆盖ramdisk中的原有文件：

```
# cp -af /home/work/busybox-1.00/_install/* /home/work/tmp
```

拷贝完毕以后，使用系统提供的映像文件制作工具mkfs.jffs2（位于CD：\software\tools），生成jffs2映像文件，命令如下：

```
# ./mkfs.jffs2 -d ./tmp -o jffs2.img
```

然后执行如下命令，卸载并压缩ramdisk.gz：

```
# umount /home/work/tmp
# gzip ramdisk
```

当以上操作全部完成以后，即可生成新的ramdisk.gz和jffs2.img文件。

注意：以上的部分操作步骤可能需要超级用户的权限。



附录 E　　系统常见问题

Q1：如何挂载一个 U 盘？

A1：U 盘的使用说明：首先插入 U 盘到系统的 USB host 接口，系统提示找到 USB 设备，然后进入/test 目录，运行命令：

$ ./udisk start

然后执行如下命令挂载 U 盘：

$ mknod /dev/sda b 8 1　　　　　；建立文件节点
$ mkdir /mnt/usb　　　　　　　　；建立挂载目录
$ mount -t vfat /dev/sda /mnt/usb　；挂载文件节点到指定目录

命令中的/mnt/usb 是新建立的 U 盘的挂载目录，进入/mnt/usb 目录就可以对 U 盘进行操作了。

Q2：如何将系统与以太网络连接？

A2：将网线插入 RJ45 口，此时系统检测到网络连接，系统的三个网络状态指示灯亮（100M 以太网），然后启动 LINUX 系统，执行如下命令为系统配置网络 IP 地址：

＃ifconfig eth0 192.168.67.56　　　　//注意该 IP 地址应在你所属的网段内

然后就可以 ping 网络中的其他计算机了。

Q3：如何使用系统的 FTP 服务器？

A3：H9200E 系统的 FTP 服务器使用步骤如下：

第一步：配置 H9200E 系统的 IP 地址

ifconfig eth0 [H9200E 系统的 ip 地址]

第二步：添加用户

adduser [用户名]

第三步：登陆 ftp 服务器

在局域网内的任意一个主机上登陆 ftp 服务器，以第二步建立的用户名和密码登陆。

ftp [H9200E 系统的 ip 地址]

```
C:\WINNT\system32\cmd.exe - ftp 192.168.67.39
Microsoft Windows 2000 [Version 5.00.2195]
(C) 版权所有 1985-2000 Microsoft Corp.

C:\>ftp 192.168.67.39
Connected to 192.168.67.39.
220 (none) FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
User (192.168.67.39:(none)): wei
331 Password required for wei.
Password:
230 User wei logged in.
ftp>
```

此时可以进行各种 FTP 操作。

Q4：如何使用系统的 Telnet 服务器？

A4： H9200E 系统的 FTP 服务器使用步骤如下：

第一步，第二步同 ftp 使用说明第一步和第二步。

第三步：登陆 ftp 服务器

在局域网内的任意一个主机上登陆 telnet 服务器，以第二步建立的用户名和密码登陆。

telnet [H9200E 系统的 ip 地址]

```
C:\WINNT\system32\cmd.exe - telnet 192.168.67.39

Linux 2.4.19-rmk7 ((none)) (0)

(none) login: wei
Password:


BusyBox v1.00 (2005.07.12-15:53+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

-sh: stty: not found
[wei@(none) wei]$
```

此时可以进行各种 Telnet 操作。